UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Bringing Probabilistic Real–Time Guarantees to the Real World

Bernardo Rabindranath Villalba Frías

Advisor

Luigi Palopoli
University of Trento

June 2018

This Ph.D. Thesis has been defended on June, 18th 2018 in front of the following evaluation committee:

Prof. Tommaso Cucinotta
Real–Time Systems Laboratory – ReTiS
S. Anna Scuola Universitaria Superiore Pisa, Italy

Prof. Giuseppe Lipari
Embedded Real–Time Adaptative system Design and Execution – Émeraude
Centre de Recherche en Informatique, Signal et Automatique – CRIStAL
Université de Lille, France

Prof. Roberto Passerone
Embedded Electronics and Computing Systems – EECS
University of Trento, Italy

A mi familia
To my family

Acknowledgements

This achievement is the result, not only of my sacrifice and effort, but it was also possible with the contributions of many different people, in different ways and places. First, I want to thank my advisor, Prof. Luigi Palopoli, for his continuous support and guidance and for sharing his time and wisdom. Your encouragement throughout this period helped me to get to this point. I would like to thank Daniele Fontanelli for his patience and the immense knowledge he offered me during my Doctoral studies. My sincere gratitude to Luca Abeni whose wide knowledge in the field was incredibly valuable throughout my whole career.

I would like to thank my labmates in the Embedded Electronics and Computing Systems group. Those who were there when I started: Federico, Alessio, Luca, Maurizio, Tizar and Payam; and those who are now that I am finishing: Paolo, Stefano, Marco, Fabiano, Valerio and Marco. Thank you for all your support both technical and scientific and for creating such a nice working environment.

To my friends at UNITN, with whom I have enjoyed pizzas, beers, lots of food and more; thank you for sharing with me your culture, ideas, points of view and even your places to crash in. Georgina, Petros and Youlie, Ligia, Arindam, Orkan, Azad and Saameh, thanks for being there for me. Thank you my friends for the company in this voyage of discovery.

A special thanks to my family in Ecuador, they were a permanent support in this challenge. To my parents, who faced endless difficulties in raising us and yet they succeeded, your sacrifice and hard–work is the greatest example. To my sisters and their families, thank you for the crazy chats, even with an intercontinental time zone in between. Thank you all for your faith and love. I love you so much.

Thanks to Cristina, my beloved wife, for being an amazing life partner. I love you amore. Thanks for encouraging me to pursue my goals, for being my support no matter what, for helping me in the difficult moments and for being the responsible of the joyful ones, for all your love and patience, for sharing and building dreams with me: *Make a plan, stick to the plan...*

And, although she made this experience a little bit more challenging, I want to thank Arianna, my sweet happy little girl. Thanks for teaching me so much in so little time, for giving me the perfect excuse for going to bed at night: *"Oggi quiero dormire col papito"* and for waking me up every morning with your wonderful smile while saying: *"Papi, per favore, posso subirme a tu camita"*. Te amo bebita.

Abstract

Stochastic analysis of real–time systems has received a remarkable attention in the past few years. In general, this analysis has been mainly focused on sets of applications competing for a shared CPU and assuming independence in the computation and inter–arrival times of the jobs composing the tasks. However, for a large class of modern real–time applications, this assumption cannot be considered realistic. Indeed, this type of applications exhibit important variations in the computation time, making the stochastic analysis not accurate enough to provide precise and tight probabilistic guarantees. Fortunately, for such applications we have verified that the computation time is more faithfully described by a Markov model. Hence, we propose a procedure based on the theory of hidden Markov models to extract the structure of the model from the observation of a number of execution traces of the application. Additionally, we show how to adapt probabilistic guarantees to a Markovian computation time. Performed over a large set of both synthetic and real robotic applications, our experimental results reveal a very good match between the theoretical findings and the ones obtained experimentally. Finally, the estimation procedure and the stochastic analysis method are integrated into the PRObabilistic deSign of Real–Time Systems (PROSIT) framework.

Keywords
Probabilistic Guarantees, Real–Time Systems, Soft Real–Time Scheduling

---

The research leading to the results presented in this thesis has been partially supported by the European Union FP7 Programme (FP7/2007–2013) under grant agreement n° ICT–2011–288917 "DALi – Devices for Assisted Living" and by the European Union's Horizon 2020 Research and Innovation Programme – Societal Challenge 1 (DG CONNECT/H) under grant agreement n° 643644 "ACANTO - A CyberphysicAl social NeTwOrk using robot friends".

Contents

Chapter 1

Introduction

Industries developing critical real–time embedded systems, such as Aerospace, Space, Automotive, Robotics, and Railways, pose stringent demands for increased processor performance to support new advanced functionality. Under this perspective, modern technology provides the opportunity to integrate multiple real–time applications, possibly of mixed–criticality levels, onto the same hardware platform.

This integration has the advantages of reducing system size, weight and power consumption, along with benefits in terms of costs reduction and reliability. However, these strategies also bring severe drawbacks regarding the feasibility to prove the correctness of the system in terms of both functional and timing/temporal behaviour.

The traditional design of real–time systems is based on three main premises: 1) each task on the system is characterised by its Minimum Inter–arrival Time (MIT) and its Worst–Case Execution Time (WCET), 2) the tasks are scheduled using static or dynamic priorities, 3) the ability for each task to meet its deadlines is guaranteed by a portfolio of algorithmic approaches, mainly focused on strict "hard" deadlines, by which a system is deemed schedulable only if every instance of every task is guaranteed to meet its deadline and even a single deadline miss is a critical failure for the application [24].

The verification of the timing correctness of a real–time system can be viewed as a two step process:

- *Timing Analysis:* aims to characterise the maximum amount of time required by the hardware platform to execute each task. In other words, the estimation of an upper bound on the WCET.
- *Schedulability Analysis:* aims to characterise the end–to–end response time of one or more tasks, considering how they are scheduled and how they interfere with each other. Generally, the schedulability analysis uses the estimated WCET to compute an upper bound on the Worst–Case Response Time.

Starting from the seminal work of Liu and Layland [72], the use of fixed or dynamic scheduling has become a common practice in the design of hard real–time systems. This design choice can be partially attributed to the possibility to estimate tight conditions for temporal guarantees of respecting the deadlines, by means of numerically efficient analysis techniques. One of the most popular techniques is the worst–case response time analysis [55] that enables the designer to estimate, for each task, the maximum delay it can possibly incur for any of its jobs.

Any analysis of this type has to account for the amount of computation time requested by an application in the worst–case, for the frequency of the job activation requests and for the delays that the execution could suffer because of the presence of other applications in the system competing for the processor: the scheduling delay.

The scheduling delay is relatively easy to account for, if the scheduler uses either static or dynamic priorities [72]. Much more critical is the knowledge of the WCET. This parameter is very difficult to measure or even to estimate when the application has strong data dependencies and/or when the architecture used for the execution is pipelined, uses a cache memory hierarchy or requires access to different shared resources. In such cases, the spread between average case and worst case could be large, and the latter could be very improbable hence escape the observation even from a large collection of execution traces.

As important as numerical analyses are analytical analyses, which establish a clear relation between the design goals (e.g., temporal guarantees) and the execution parameters of the task. The most famous analytical approach for single–processor scheduling algorithms is the Utilization Bound [72], which offers clear guidelines on how to tweak periods and computation times in order to meet the deadlines of all tasks in the system.

In the same line of reasoning, the Time Demand Analysis (TDA) [69] provides a more accurate insight on the ability of a fixed–priority single–processor system to meet all the deadlines. The idea is very simple: in any interval, the computation time demanded by all tasks in the set must never exceed the available time. It is based upon the observation that the worst–case response time of a job occurs when it is released at a critical instant, namely when a job in each task is released along with a job from all tasks of equal or higher priority.

The whole point of these analytical bounds is to provide a clear perception on the design goals and on the impact of the design parameters in their attainment. As an example, using the utilisation bound, a designer knows how to tweak the task activation periods to achieve a schedulable set, and this is far more valuable than a simple yes/no answer. This "sensitivity" analysis can be carried out in different and more accurate ways [22, 21, 23], but the clarity and the "physical" insight of simple analytical tests

remains unmatched.

Unfortunately, classic real–time scheduling algorithms are known to be unfit for soft real–time systems, a class of real–time applications that are resilient to occasional and controlled timing faults and for which the strict respect of every real–time constraint can be easily traded for a more efficient management of the system resources. This is achieved at the cost of a degradation in the provided Quality of Service (QoS). Such a degradation often depends on the number and severity of the constraint violations over a certain interval of time.

As an example, for a video on–demand system, occasional losses or delays of frames can be very hard to perceive even for an experienced user. Only when the occurrence of these anomalies becomes too frequent or uncontrolled does the QoS degrade (producing annoying glitches or "stop–and–go" effects). As surprising as it may seem, the same applies to many robotic control applications [45, 27].

The market penetration of soft real–time applications is so relevant that it has stimulated an intense research activity aiming for alternative scheduling solutions for soft real–time systems. Coupled with these scheduling solutions, it is also necessary to develop numerical or analytical analysis techniques that allow the designer to guarantee the temporal behaviour of the system.

In the last years, probabilistic design has emerged as a viable option for soft real–time systems. In this scenario, the system designer is required to reason about the impact of her scheduling choices on the performance of the system in stochastic terms: the computation requirements of a task as well as the system performance can be described by probability distributions. In order to tackle this issue, a number of stochastic or probabilistic analysis for these systems have been proposed.

However, the vast majority of the models proposed so far for stochastic analysis of soft real–time systems, make the assumption that the computation time of the task is independent and identically distributed (i.i.d). While this assumption does not impair the application of the analysis in many cases of interest, this may not be true for many robotic applications.

For instance, for the computer vision applications that mobile robots use to sense the environment, it can be argued that the computation workload depends on the complexity of the scene. Pictures shot in close sequence are likely to require the same computation time, which can change in magnitude when the robot moves towards an emptier area. This effect introduces a potentially strong correlation in the stochastic process describing the computation time, reducing the appeal of techniques developed for i.i.d. processes.

1.1 Motivation

A large class of modern soft real–time robotic applications uses randomised methods at the inner core of its functionality. The use of these methods leads to wide variations in the computation time discouraging the use of standard hard real–time analysis techniques. Additionally, probabilistic guarantees for these systems have been developed based on the traditional assumption that the inter–arrival and computation times are described by an i.i.d. stochastic processes.

However, this assumption cannot be considered realistic for many cases of interest. Indeed, several robotic applications exhibit a strong correlation structure in their computation times depending on the different conditions in which the robot operates. This violation in the main assumption could jeopardize the stochastic analysis, reducing its accuracy and limiting its ability to provide probabilistic guarantees in a soft real–time system.

Given the rapidly expansion of real–time robotic applications, the increased use of randomized algorithms and the need for accurate and tight probabilistic guarantees of respecting the deadline for such applications, we contribute to this direction by investigating usable numeric techniques for the stochastic analysis and design of soft real–time systems, scheduled by a resource reservation algorithm and presenting dependencies in their computation times.

We strongly believe that this first step towards the development of such techniques will create important opportunities to optimise the design of robotic systems and to reduce their costs.

1.2 Related Work

The stochastic analysis of the performance of soft real–time tasks started two decades ago and, in the last few years, it has received a remarkable attention. An important number of research papers focused on probabilistic methods for schedulability analysis of systems with fixed–priority, by extending existing response time analysis and substituting the fixed computation times with random variables. Under this scenario, Probabilistic Time Demand Analysis (PTDA) [96] is presented. This approach uses convolutions of probability functions of the computation times to obtain the probability function of the response time of a task; however, the deadlines cannot be greater than the periods.

The Statistical Rate Monotonic approach [11] assumes harmonic periods for the tasks and proposes a variation of the Rate Monotonic scheduler [72] implementing a budgeting mechanism and dropping jobs that cannot be completed in time: a job is allowed to start

only if the budget is sufficient to ensure its completion and all other jobs are dropped.

Stochastic Time Demand Analysis [49] extends PTDA to include systems where the deadlines may be greater than the period. This approach focuses on providing a statistical measure of the amount of missed deadlines to be expected. Both analyses are based on the sum of random variables, and thus the use of convolutions to determine their probability function. However, the analyses cannot address systems where the maximum system utilization is greater than one.

The application of Extreme Value statistical analysis to model the tail of the distributions is used in [41]. This analysis uses end–to–end measurements for the computation time of a task to reason about the probability of the WCET of being greater than the largest computation time observed during any of the runs of the program. A similar approach is followed in [14], where, additionally to the data obtained from measurements, a static analysis of small sections of the code is performed. The authors reason on the WCET of the whole program by combining probabilistically the worst–case effects of the individual sections. However, these approaches based on the combination of the worst–case situation of every step in the analysis will lead to a considerable pessimistic assumption in the behaviour of the system which could be problematic for soft real–time systems.

More recently, an important number of research papers focused on the computation of the response time of systems with fixed or dynamic priority when tasks have stochastic variability in the computation times [38, 60]. The presented technique is based on Markov processes, allowing the authors to reason probabilistically about the probability of the system to respect the deadline. It provides both analytical and numerical solutions for the deadline miss probabilities of the tasks by computing the complete probability function of the response time of each task. Besides, other studies on the computation of the response time of systems where the tasks have stochastic variability in the computation times [78] has been presented.

Furthermore, the analysis of tasks presenting stochastic variability in the inter–arrival time has been also considered [36]. However, the computation times of the tasks are described by their worst–case value. An hybrid approach is presented in [56], where both: the computation and the inter–arrival times of the tasks are given by random variables with known distributions.

Similar techniques have been recently applied to multi–processor systems [79, 80]. The authors showed that, allocating the processing resources based on the average computation time, the tardiness and the response time of a task is bounded. The authors also offer a very conservative result on the probability of deadline miss, which is applicable only if deadlines are much larger than the period.

Beside the traditional approaches based on fixed or dynamic priorities, different authors have analysed other types of scheduling approaches. For instance, in [57] a Time Division Multiple Access (TDMA) approach that shares resources among tasks in a distributed system is analysed. In [54] it is proposed an approach that divides each job into a mandatory part and a number of optional parts. The system schedules the jobs in such a way that it provides deterministic guarantees for the mandatory parts and probabilistic guarantees for the optional parts of the jobs.

Additionally, the concept of probabilistic Worst–Case Execution Time (pWCET) [15] has been presented. The pWCET can be used when the absolute WCET cannot be precisely determined. Instead of assuming a single value for the WCET, the pWCET represents a probability distribution of the WCET. The idea is that for each execution, a task experiences a WCET, which changes depending on the input data set and on random effects in the system hardware (e.g., due to the presence of a cache). Therefore, it is possible to think of the WCET as a random variable called pWCET.

The probabilistic analysis based on pWCET can be classified into two main categories:

- *Static Probabilistic Timing Analysis (SPTA)* [9, 71]: is applicable when some part of the system or its environment contributes with random or probabilistic timing behaviour. SPTA methods derive the pWCET distribution for a program by analysing the structure of the program and modelling the behaviour of the hardware it runs on. Note that SPTA does not execute the code on the actual hardware; rather it relies on the correctness of the hardware model.

- *Measurement–Based Probabilistic Timing Analysis (MBPTA)* [41, 35, 92]: makes use of measurements of the overall computation time of the tasks, obtained by running them on the actual hardware. Rather than taking the maximum observed computation time and then adding some engineering margin, these methods use statistical techniques grounded in the Extreme Value Theory (EVT) to estimate the pWCET distribution.

This two type of analysis originate two different meanings for the pWCET distribution:

- The pWCET distribution obtained from SPTA is a tight upper bound on all of the probabilistic computation time distributions that could be obtained for each individual combination of inputs and hardware/software states, excluding the random variables which give rise to variation in the timing behaviour.

- The pWCET distribution obtained from MBPTA is a statistical estimate giving an upper bound p on the probability that the computation time of a task will be greater than some arbitrary value x, valid for any possible distribution of input values that could occur during deployment.

In the case of pWCET distributions obtained from SPTA the key idea is to conservatively model the execution times of jobs as independent random variables, allowing the designer to use simple convolution to derive probabilistic Worst–Case Response Time distributions [38, 78], which can then be compared to the appropriate deadline to determine the probability of a deadline miss.

On the other hand, the pWCET distributions obtained from MBPTA do not give the designer any information about the probability that the execution time of any particular job exceeds some value x, but rather provide a statistical estimate of the probability that the maximum computation time of a task, over a large number of jobs, exceeds x in some operating cycle. Hence, it is not possible to use basic convolution since that would assume independence of job computation times that typically does not exist.

When dealing with access to shared resources or multi–processor architectures, the presence of interactions either across cores or through shared hardware (e.g., cache, bus, DMAs, etc.) is unavoidable. These interactions are difficult to characterise, leading to a high pessimism in the estimation of the WCET.

Several techniques, mainly at hardware level, have been proposed to address different sources of unpredictability including real–time handling of peripheral drivers, real–time compilation and analysis of contention for memory and buses. The PREdictable Execution Model (PREM) [88, 101] uses scheduling to reduce or eliminate contention for shared resource accesses.

Regarding to cache–management, two techniques: *cache locking* and *cache scheduling* are presented in [100]. Under cache locking, portions of the cache are viewed as non–preemptive resources that are accessible via a locking protocol. Under cache scheduling, portions of the cache are viewed as preemptive resources that are "scheduled".

Moreover, operating–system (OS) infrastructure that allows mixed criticality applications to be supported on a multi–core platform was proposed in [10]; the proposed ideas were implemented as the Mixed Criticality on MultiCore (MC^2) [61] framework.

For soft real–time applications, the traditional notion of "deterministic" deadline is not expressive enough to formulate the QoS requirements of the application. A very natural direction is offered by the *probabilistic deadlines* [3]: a deadline is associated with a probability of meeting it, which in turn can be related to the QoS delivered by the application [62, 45] and, more generally, enable the expression of a wide range of performance requirements, where classic hard real–time systems can be regarded as a special case. It can be argued that the QoS experienced by several types of industrial applications can be related to such probabilistic performance metrics.

A probabilistic real–time guarantee states that the deadline will be respected with a probability at least equal to the one specified in the probabilistic deadline. A probabilistic

guarantee of this kind requires a system analysis and solution techniques based on the real–time queueing theory [68]. Such tools allow the computation of the response time distributions of a set of tasks for different scheduling algorithms when the system is under heavy traffic conditions, namely systems with a workload of the tasks very close to 100%, which restricts the practical applicability of those results.

The analysis turns out to be much simpler if resource reservation scheduling [90, 1] is adopted, which enables a fine grained control on the fraction of computing power (bandwidth) that each task receives. A key property of resource reservation scheduling is the so called *temporal isolation*: the ability for a task to meet its deadlines solely depends on its computation requirement and on its scheduling parameters.

This property introduces a drastic simplification in system design and is probably the main reason for the increasing popularity of resource reservations. It allows the system designer to model the evolution of a soft real–time task as a Discrete–Time Markov Chain (DTMC) with an infinite number of states [3, 4]. Moreover, this DTMC takes the form of a Quasi–Birth–Death Process (QBDP), hence computing the steady state probability of respecting the deadline can be efficiently achieved by one of the several numeric solutions for QBDP [81, 67, 18] that can be found in the literature.

By taking advantage of this model, different authors have analysed the stochastic behaviour of reservation–based schedulers using numeric techniques [8, 75] or developing analytical bounds [85]. Such solutions strike different trade–offs between performance and accuracy. For instance, when designing a complex system composed by many applications, analytic techniques are preferable since they ensure a quick convergence to a sub–optimal solution. If resources are scant, it can be convenient to seek a more accurate solution paying the price of a longer design time.

When using reservation–based schedulers, the traditional approach consists on reserving a fixed fraction of the CPU bandwidth to each task. However, it can be argued that the static allocation of CPU is not efficient enough when dealing with wide variations in the computation times experienced by the tasks. This problem can be addressed by dynamically adapting the amount of resources reserved to each task: adaptive reservations [2].

Several algorithms have been proposed in the literature to implement adaptive reservations. For instance, a non linear feedback control scheme that adjusts the bandwidth based on two factors: a prediction of the possible range of variation of the next computation time and the computation time of the previous job is presented in [83]. Formal proof of stability in the stochastic sense for generic families of controllers is provided in [33].

In the domain of stochastic control, the Stochastic Dead Beat [34, 31] was introduced. The goal is to choose a bandwidth such that the expectation of the next scheduling error

be equal to zero. This expectation is conditioned to the past evolution of the system. At each step, the controller tries to optimise the expected values of certain quantities of interest based on the expected behaviour of the computation time stochastic process. The presented architecture includes a separate component, the predictor, which is responsible for providing the necessary information based on its knowledge of the past evolution of the system.

A customary assumption made in the literature on stochastic analysis of real–time systems is that the inter–arrival and computation times are independent and identically distributed (i.i.d.) stochastic processes. However this assumption is not correct for several real applications. For this reason, different authors have investigated the effects of the dependencies in the computation times on the WCET and schedulability analyses, and questioned the applicability of the i.i.d. assumption in the area of real–time applications [93]. An i.i.d. process makes the system's analysis tractable [26] and can be considered as a good approximation in some cases. However, there are also practical cases in which ignoring the correlation structure could lead to significant errors in the analysis.

In the case of WCET analysis, the use of copulas to provide distributional bounds is described in [13], where perfect dependence is a good assumption because computation times of different program blocks can depend on common parameter settings. This idea could be slightly modified in such a way that it allows us to estimate an i.i.d. over–approximation of the computation times starting from non–independent ones. A similar idea is presented in [73], where the authors tackle the correlation problem decomposing the computation times in two parts: a dependent part which is upper bounded by a deterministic independence threshold and an additional i.i.d. component.

In order to make the stochastic analysis of real–time systems accessible to a large class of interested designers, several software tools for the simulation analysis and design of the schedulability of real–time systems have been proposed. It is worth to mention MAST [51], a tool set for modelling the temporal and logical components of real–time systems and for performing real–time scheduling analysis on the model. Cheddar [95] is a framework designed to check task temporal constraints which provides two features: feasibility tests and a scheduling simulation engine.

RTSim [86] is a software tool developed for the co–simulation of real–time embedded controllers and plants in order to evaluate the timing properties of the architecture in terms of control performance. It consists of several components including schedulers, task models, etc. It supports both probabilistic computation and inter–arrival times.

Additionally, STORM [97] is a multi–processor simulation tool for scheduling evaluation, which also measures other features such as energy consumption and CPU load. YARTISS [28] is an open source simulator aimed at the evaluation of real–time scheduling

policies on multi–processor systems for various models of tasks. SimSo [29] is a simulator designed for the comparison and the understanding of real–time multi–processor scheduling algorithms.

As far as tools for probabilistic design are concerned, it is possible to mention the existence of PAnSim [76], an analysis and simulation tool which currently supports fixed–priority preemptive scheduling. The tool is based on the notion of pWCET and probabilistic Minimal Inter–arrival Time (pMIT) distributions, nevertheless it also supports probabilistic and arbitrary deadlines.

In the same line of work, we can mention Probabilistic SimSo [76], which is a probabilistic extension of SimSo [29] with support for pWCET and pMIT, and DIAGXTRM [53]. The latter implements the Measurement–Based Probabilistic Timing Analysis (MBPTA), which is an application of the Generalized Extreme Value Theory [92] to estimate the pWCET and evaluate its quality.

1.3 Objectives

The scope of this thesis is to explore a set of techniques aimed at addressing the problem of providing accurate probabilistic guarantees for soft real–time applications which are characterised by computation times that present a strong correlation while executing on single–processor systems.

The specific objectives of this work are:

- To investigate state–of–the–art approaches for probabilistic analysis of soft real–time systems where the computation times are described both by i.i.d. and non i.i.d stochastic processes.
- To propose a more faithful model to describe the computation times of a class of robotics applications which are not independent and identically distributed.
- To adapt the available solutions for probabilistic guarantees on single–processor systems to the scenarios where the computation times can be described by the proposed model.
- To develop methods for the estimation and validation of the parameters characterising the proposed model from the experimentally collected data.
- To perform evaluations and analyse the behaviour of the proposed methods under different real robotic scenarios.

- To explore both simulations and real–life data in the different evaluation stages, in order to verify the applicability of the proposed solutions in real environments and conditions.
- To develop a software tool that implements both the probabilistic analysis and the parameter identification for a real–time task characterised by the proposed model.

1.4 Contributions

The specific contributions of this work are:

- The identification of a Markovian model for the characterisation of the computation times of a large class of robotic applications.
- An effective procedure for identifying the different modes, the transition probability among modes and the distribution of the random variable in each mode of the proposed Markovian model. This procedure uses the theory and adapts the techniques developed for hidden Markov models.
- The extension of the probabilistic guarantees for resource reservations, initially developed for i.i.d. stochastic processes, to the case of the proposed Markovian computation time.
- The proposed analysis techniques are implemented on a software tool, called the PROSIT tool, that facilitates the access to the probabilistic analysis of soft real–time systems for a potentially large number of researchers and industrial practitioners. PROSIT[1] is implemented as a C++ library under the GNU GPL license and is available online including several examples and datasets.
- The dissemination of the proposed methods and experimental results:
  - *Luigi Palopoli, Danielle Fontanelli, Luca Abeni, and Bernardo Villalba Frías. "An analytical solution for probabilistic guarantees of reservation based soft real–time systems". IEEE Transactions on Parallel and Distributed Systems, 27(3):640–653, March 2016.*
  - *Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. "Probabilistic real–time guarantees: There is life beyond the i.i.d. assumption (outstanding paper). In Real–Time and Embedded Technology and Applications Symposium (RTAS), pages 175–186, Pittsburgh, USA, April 2017. IEEE.*

[1] https://bitbucket.org/luigipalopoli/prositool.git

- *Luca Abeni, Daniele Fontanelli, Luigi Palopoli, and Bernardo Villalba Frías. "A Markovian model for the computation time of real–time applications". In Instrumentation and Measurement Technology Conference (I2MTC), 2017 IEEE International, pages 1–6. IEEE, 2017.*
- *Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. "The PROSIT tool: Towards the optimal design of probabilistic soft real–time systems", accepted for publication in Software: Practice and Experience, 2018.*

1.5 Thesis Outline

The remaining part of this document is organized as follows. Chapter 2 defines the main components of the probabilistic guarantees framework that will be useful throughout the thesis.

Chapter 3 provides basic definitions on Random Events and Markov chains. Additionally, it is presented how a task scheduled by a resource reservation can be conveniently modelled as a QBDP and how to derive a conservative approximation of this model, which has a parametric accuracy and which retains the structure of a QBDP.

Chapter 4 summarizes two existing strategies applied to the problem of probabilistic guarantees for computation times described by an independent and identically distributed stochastic process. In particular, it shows an efficient numeric solution for the steady state probability of respecting the deadline. Moreover, after introducing an additional simplification in the model, it introduces a closed form conservative bound, which proved itself reasonably accurate for the test cases.

The main contributions of this research are presented in Chapter 5, where the Markov Computation Time Model (MCTM), a Markovian model of the computation time for robotic applications that exhibit a strong correlation structure given by the robot operating conditions, is introduced. Moreover, it shows a technique for the extraction of the parameters of the MCTM from a limited number of observation and describes how to adapt the techniques developed for probabilistic guarantees of resource reservations to MCTM computation times.

Chapter 6 introduces an extensible and open source design tool, called PROSIT[2], which enables the probabilistic analysis of the temporal performance of a real–time task under fixed–priority and resource reservations scheduling algorithms. For resource reservations, the tool also offers an automatic procedure for the synthesis of scheduling parameters that optimise a quality metric related to the probabilistic behaviour of the tasks. Finally, the conclusions and future directions of this research are drawn in Chapter 7.

[2]https://bitbucket.org/luigipalopoli/prositool.git

Chapter 2

Definitions on Real–Time Scheduling

The computation activities within a computer are executed by pieces of software called *tasks*. These tasks share the *Central Processing Unit* (CPU), cooperating or competing among each other. When the application is composed of only one task, it is called single–task application, while multi–task applications are composed of multiple interacting and cooperative tasks.

Generally speaking, a real–time task τ_i, with $i \in \mathbb{Z}_{>0}$, is a cyclic piece of software (e.g., a process, a thread) with an associated temporal constraint, called *deadline*, that must be respected to ensure the correctness of the application. Upon each activation request, the application executes an instance of the task, called *job*. The job acquires the new input (if any), updates the internal state (if any) and generates an output.

Therefore, real–time tasks consist of a sequence, possibly infinite, of jobs. A periodic task is a task which is periodically activated after a fixed amount of time, called *task period* (T_i). For aperiodic tasks, the interval of time between two jobs can be different.

If missing a deadline causes a critical failure in the system then the deadline is said to be *hard*. A deadline is said to be *soft*, if a deadline miss causes a degradation in the Quality of Service (QoS), but is not a catastrophic event.

2.1 Application Model

Single–task applications consider a set of independent applications, each one composed of a single real–time task. The applications share the same computing resources, but do not interact in other ways.

From the temporal point of view, the j^{th} job of the i^{th} real–time task, $J_{i,j}$, is characterised by the triplet ($a_{i,j}$, $c_{i,j}$, D_i), as presented in Figure 2.1. The three parameters are described as follows:

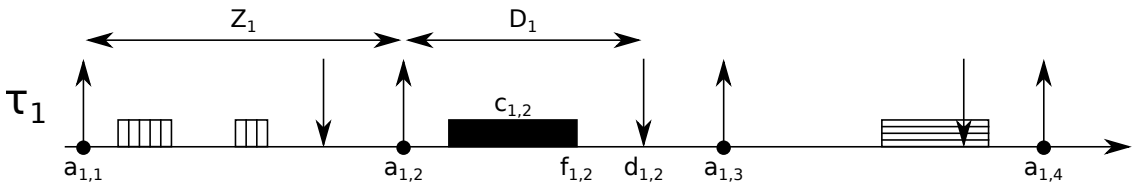

Figure 2.1: Mathematical model of a periodic real–time task τ_1 with period Z_1 and relative deadline D_1. The three patterns denote different jobs. The second job (black rectangle) is activated at time $a_{1,2}$, it executes for a computation time $c_{1,2}$ and it finishes its execution at time $f_{1,2}$. The job respects its deadline since $d_{1,2} > f_{1,2}$. On the other hand, the third job (horizontal lines) misses its deadline.

- $a_{i,j}$ is the activation instant of the job. Generally speaking it is possible to write: $a_{i,j} = a_{i,j-1} + Z_i$, where Z_i represents the inter–arrival time of the task. For periodic tasks, the inter–arrival time is the task period, $Z_i = T_i$; while, for aperiodic tasks, Z_i is a stochastic process with Probability Mass Function (PMF) $\mathcal{Z}_i(z) = \mathbf{Pr}\{a_{i,j} - a_{i,j-1} = z\}$;
- $c_{i,j}$ is the computation time of the j^{th} job, which is also assumed to be a stochastic process $\mathcal{C}_i$ with known distributions;
- D_i is the relative deadline, meaning that the j^{th} job is required to finish within $d_{i,j} = a_{i,j} + D_i$. More formally, defining the finishing time, $f_{i,j}$, as the instant where the job completes its execution, the job has met its deadline if $f_{i,j} \leq d_{i,j}$ and has missed it otherwise: $f_{i,j} > d_{i,j}$. Additionally, the *response time* of the job, $r_{i,j}$, defined as the interval between the activation and the finishing time, is given by:

$$r_{i,j} = f_{i,j} - a_{i,j}. \tag{2.1}$$

  In this case, the deadline is met if $r_{i,j} \leq D_i$.

2.2 Model of the Computation Time

Most of the existing literature on probabilistic analysis of real–time systems assume that the computation time of each job, $c_{i,j}$, is described by an independent and identically distributed (i.i.d.) stochastic process characterised by a Probability Mass Function (PMF) $\mathcal{C}_i(c) = \mathbf{Pr}\{c_{i,j} = c\}$.

The choice of a PMF over a Probability Distribution Function (PDF) is related to the fact that the computation times can only take values that are multiples of the processor

clock, hence they are discrete in nature. However, modern computing systems have increased their time granularity up to the nanosecond resolution, allowing the designers to describe the computation time as a continuous stochastic process. In this thesis, the computation time is assumed to be described by a PMF, following the traditional approach. However, the proposed analysis can also be performed considering a PDF to describe the computation times.

Although the i.i.d. assumption can be considered valid for several real–time applications, there are many other cases where is not necessary true. For instance, consider a computer vision application used by a mobile robot to sense the environment, the pictures shot in close sequence are likely to be related among each other. This effect introduces a potentially strong correlation in the stochastic process describing the computation time, reducing the appeal of techniques developed for i.i.d. processes.

In other words, the computation time of a job is correlated to the computation times of previous jobs, hence it cannot be described using a simple probability distribution function $\mathcal{C}_i(c)$, as done in the vast majority of the literature. As a consequence, more advanced mathematical techniques have to be used to properly and more precisely describe the computation times.

Therefore, in this thesis it is not assumed that the stochastic process $\mathcal{C}_i$, modelling the computation time of the real–time tasks, is an independent and identically distributed (i.i.d.) process. On the contrary, its computation time is allowed to be a Markov Modulated stochastic Process [43] (MMP). This model can be described in the following terms: the task τ_i switches between N different operating modes, with the switches ruled by a Markov chain; and, in each of these modes, the computation time behaves as an i.i.d. process.

More formally, this model is described by a triplet:

$$\{\mathcal{M} = \{m_1, \ldots, m_N\}, \mathcal{P} = (p_{a,b}), \mathcal{C} = \{C_k\}\},$$

where $\mathcal{M}$ is the set of all possible modes, the matrix $\mathcal{P}$ is the mode transition probability from m_a to m_b, where m_a, $m_b \in \mathcal{M}$, and $\mathcal{C}$ is the set of distributions characterising the computation time in each mode m_k, where $m_k \in \mathcal{M}$.

This model has been defined Markov Computation Time Model (MCTM) [98] and has been shown to be a good fit for many real–life applications. It is important to point out that, when a MCTM is composed of a single mode (i.e., a single state), the stochastic process modelling the computation time reduces to a standard i.i.d. process.

2.3 Probabilistic Deadline

For decades, the analysis of real–time systems was based on the strict requirement that all the deadlines have to be respected: the notion of *hard deadlines*. However, in the past few years, soft real–time systems have experienced an undisputed growth in embedded systems. A soft real–time application is one for which the deadlines can occasionally be missed, but the probability of this event has to be controllable and predictable.

For these applications, the traditional notion of hard deadlines is insufficient per–se to formulate their QoS requirements. Hence, designers of soft real–time systems need effective means to reason about the impact of their scheduling choices on the performance of the system in stochastic terms.

An effective method to express the timing requirements for a soft real–time application is by associating each deadline with a probability that it will be met: the notion of *probabilistic deadlines* [3]. A probabilistic deadline is a pair (D_i, γ_i), where γ_i is the steady state probability of respecting the relative deadline D_i.

$$\gamma_i = \mathbf{Pr}\left\{r_{i,j} \leq D_i\right\}, \tag{2.2}$$

where $r_{i,j}$ is the response time as defined in Equation (2.1).

Generally speaking, the designer can be interested in knowing the probability of meeting the deadline for different values of D_i. Thereby, the notion of Quality of Service adopted in this thesis is essentially related to the frequency of deadline misses or, more formally, to the distribution of the response times.

The probabilistic deadline is respected if $\mathbf{Pr}\left\{f_{i,j} > a_{i,j} + D_i\right\} \leq \gamma_i$. If $\gamma_i = 0$, the deadline is considered hard. Additionally, it is possible to specify a sequence of h probabilistic deadlines for τ_i:

$$\left[\left(D_i^{(1)}, \gamma_i^{(1)}\right), \ldots, \left(D_i^{(h)}, \gamma_i^{(h)}\right)\right],$$

where $\gamma_i^{(x)} \geq \gamma_i^{(y)}$ if $D_i^{(x)} > D_i^{(y)}$, formulating in this way, a specification on the distribution of the delays.

2.4 Application Quality

While probabilistic deadlines effectively capture the real–time constraints, the quality perceived by the user of the application is not immediately expressed by such quantities. Much more useful is a quality index μ_i, which is necessarily application–specific and is related to the design choices.

This index is related to the probabilistic deadlines by a functional dependence. In the simplest case, it is possible to consider a single probabilistic deadline to be equal to the relative deadline D_i. In this case, the index μ_i is a monotone non–decreasing function of probability γ_i. More generally, if a sequence of h probabilistic deadlines is considered, then μ_i is a function of the vector $\left[\gamma_i^{(1)}, \ldots, \gamma_i^{(h)}\right]$.

The notion of *Quality function* used in this thesis is inspired to the QRAM framework [91]. However, QRAM assumes a functional dependence between the scheduling parameters and μ_i, which can be difficult to identify. On the contrary, we relate the quality to the probabilistic temporal behaviour of the task, which depends on the scheduling parameters in non–obvious ways. This choice allows the designer to reason about the system quality in a natural conceptual framework without committing to any particular scheduling policy.

For instance, in the domain of control applications, a quality index of this kind could be the steady state covariance of the controlled plant state, which is known to be a function of the distribution of the delays [45].

Likewise, for a media processing application, we can define μ_i as the Peak Signal–to–Noise Ratio (PSNR) or the Structural Similarity Index (SSIM) between the original media and the one reproduced by the application [63]. Under the assumption that the video frames decoded by jobs that finish late are not displayed, when a deadline is missed, the PSNR or SSIM is computed using the last frame decoded in time [62] and degrades with the frequency of this event.

2.5 Scheduling Algorithm

For the class of real–time applications considered in this thesis, whose quality of output depends on sufficient access to a resource over time, a particular type of scheduling algorithm called *resource reservations* has emerged as the most suitable choice. Reservation techniques have proved to be very effective in providing temporal isolation, meaning that when a task has reserved a specific amount of a resource, it must have access to that reserved amount of the resource regardless of other tasks running on the system. Moreover, these techniques have been implemented in a number of different systems using different scheduling algorithms [1].

A reservation is a pair (Q_i^s, T_i^s)[1], where Q_i^s is the amount of time that the task is guaranteed to use the resource within every reservation period T_i^s. The fraction of resource utilization dedicated to task τ_i is given by $B_i = Q_i^s/T_i^s$ and it is usually called *bandwidth*.

[1]The s superscript stands for "server" and it means that the two parameters are associated with the server, while the i subscript refers to the task being served.

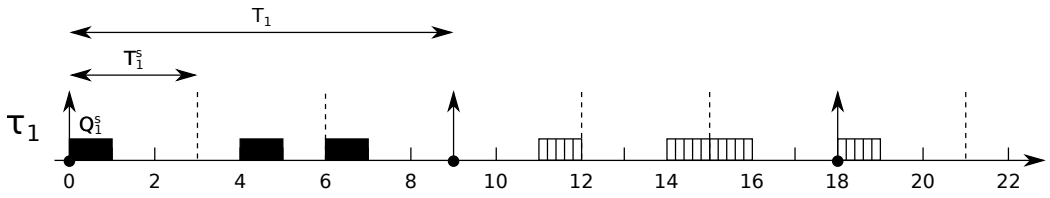

Figure 2.2: Schematic of a periodic real–time task τ_1 with period T_1 scheduled by a resource reservation algorithm. The two patterns denote different jobs. The scheduling parameters are: reservation period $T_1^s = 3$ time units and budget $Q_1^s = 1$ time units, meaning that the task is guaranteed to execute for a maximum of 1 time unit every 3 time units.

Q_i^s is also called the *budget* or capacity of the reservation while T_i^s is called the *server period*. Figure 2.2 shows a periodic task with period equal to the deadline $T_i = d_i = 9$ time units and reservation parameters $Q_i^s = 1$ and $T_i^s = 3$ time units.

The behaviour of a reservation is the following: a task τ_i attached to a reservation (Q_i^s, T_i^s) receives the guaranteed Q_i^s time units in every reservation period T_i^s. If a task tries to execute for more than the guaranteed bandwidth, the scheduler throttles it, avoiding the task to be selected for execution, until the end of the current reservation period. As a result, each task is limited to use a maximum of Q_i^s every T_i^s units of time.

Although any scheduling algorithm could be used to implement a reservation strategy, the use of a dynamic–priority scheduler permits to obtain a more efficient implementation. The particular implementation of the resource reservations approach considered in this thesis is the Constant Bandwidth Server (CBS) [1, 70].

For each task, the scheduler manages two state variables: the *remaining runtime* (q_i), representing the amount of remaining processor time that the task can use in the current reservation period; and the *scheduling deadline* $(d_{i,j}^s)$, which is used to assign a dynamic priority to the task. When the CBS starts handling a task, q_i and $d_{i,j}^s$ are initialized to 0. In the CBS, reservations are implemented by means of an EDF scheduler.

EDF schedules the tasks based on their scheduling deadlines $d_{i,j}^s$, which are dynamically managed by the CBS algorithm. When a new job $J_{i,j}$ arrives, the server checks whether it can be scheduled using the last assigned scheduling deadline $d_{i,j-1}^s$ and the remaining budget q_i. In the affirmative case, the scheduling deadline of the job is initially set to the current deadline $d_{i,j}^s = d_{i,j-1}^s$ without updating the budget. Otherwise, the scheduling deadline $d_{i,j}^s$ is set equal to $a_{i,j} + T_i^s$ and the budget is set equal to $q_i = Q_i^s$.

Every time the job executes, its remaining budget q_i is decreased consequently. When the remaining budget q_i arrives to 0, the task τ_i is throttled and cannot be selected by the scheduler for execution. Therefore, the task τ_i is not schedulable until time $d_{i,j}^s$, when the

runtime will be replenished to its maximum value ($q_i = Q_i$) and the scheduling deadline will be postponed ($d^s_{i,j} = a_{i,j} + T^s_i$).

The scheduling deadline $d^s_{i,j}$ has, in general, nothing to do with the deadline $d_{i,j}$ of the job: it is simply instrumental to the implementation of the CBS.

As a consequence, each task is reserved an amount of computation time Q^s_i in each server period T^s_i regardless of the behaviour of the other tasks, a property called *temporal isolation*. The EDF scheduler is able to provide schedulability guarantees to a set of real–time tasks if their total utilisation is smaller than a certain threshold. For instance, on single–processor systems (or on multi–processor systems using partitioned scheduling), EDF guarantees schedulability if and only if:

$$\sum_i B_i = \sum_i \frac{Q^s_i}{T^s_i} \leq 1. \tag{2.3}$$

The temporal isolation property is of the greatest importance for the probabilistic analysis since it allows us to assign a bandwidth to each task. As a consequence, the task executes as if on a dedicated processor and the analysis of the scheduling delays is significantly simplified. Therefore, it is possible to remove the subscript i meaning that the analysis refers to one specific task.

Note that the temporal isolation provided by the CBS is related to processor scheduling. In a complex multi–core environment or in presence of shared resources, other types of interference between the execution of the tasks could come from conflicts on the memory bus or from cache–related delays. In these cases it is important to consider, when measuring the computation times, the blocking time related to the mutual exclusion access to the shared resources.

The probabilistic analysis presented in this thesis does not consider any hardware effect, which are managed at a system level. In fact, the analysis assumes that the computation times of the tasks are described by a PMF, which should include all the possible additional time required to handle possible interferences. As long as the observed computation times respect such assumption, the obtained probabilities of respecting the deadline can be considered valid.

Moreover, it is worth to point out that reservation–based schedulers are now a commonplace technology distributed with main stream Linux kernels under the name of SCHED_DEADLINE [70]. The current implementation in the Linux kernel improves the original CBS by allowing to specify a relative deadline D_i different from the period T_i.

Additionally, SCHED_DEADLINE deals with multi–processor scheduling in a similar way that *distributed runqueue* in Linux. Specifically, it implements one ready queue for each processor, as in partitioned scheduling, along with the logics for migrating tasks

among the different ready queues, as in global scheduling. Therefore, SCHED_DEADLINE supports both multi–processor scheduling schemes: partitioned and global scheduling.

Chapter 3

Background on Stochastic Analysis

In this chapter, since the tasks parameters will be modelled as stochastic processes, some basic definitions from the theory of random processes are introduced. Additionally, some basic definitions on Markov chains and in particular on Quasi–Birth–Death Processes (QBDP) will be presented along with the detailed modeling, as a QBDP, of a task scheduled by a resource reservation algorithm. Finally, how to derive a conservative approximation of this model, which has a parametric accuracy while retaining the structure of a QBDP, is described.

3.1 Random Events

A *random experiment* is any experiment whose outcome is uncertain. The set of all possible outcomes of a random experiment is called *sample space* and is denoted by Ω. An event is a possible outcome of a random experiment and is associated to a subset of Ω. The set of all possible events is called *event space* and is denoted by $\mathbb{E}$.

The pair $(\Omega, \mathbb{E})$ is called *probability space*. Given a probability space $(\Omega, \mathbb{E})$, the probability $\mathbf{Pr}\{.\}$ is a function from $\mathbb{E} \to \mathbb{R}$ such that:

- $\mathbf{Pr}\{\mathbb{A}\} > 0$, $\forall \mathbb{A} \in \mathbb{E}$;
- $\mathbf{Pr}\{\Omega\} = 1$;
- If the events $\mathbb{A}_1$, $\mathbb{A}_2$, ..., $\mathbb{A}_n$ are disjoint, then $\mathbf{Pr}\{\bigcup_{i=1}^{n} \mathbb{A}_i\} = \sum_{i=1}^{n} \mathbf{Pr}\{\mathbb{A}_i\}$.

For a generic couple of events $\mathbb{A}_1$ and $\mathbb{A}_2$, they are called independent if the occurrence of one does not affect the probability occurrence of the other. More formally, $\mathbb{A}_1$ and $\mathbb{A}_2$ are independent if $\mathbf{Pr}\{\mathbb{A}_1 \cap \mathbb{A}_2\} = \mathbf{Pr}\{\mathbb{A}_1\} \cdot \mathbf{Pr}\{\mathbb{A}_2\}$. The conditional probability is defined as:

$$\mathbf{Pr}\{\mathbb{A}_1 \,|\mathbb{A}_2\} = \frac{\mathbf{Pr}\{\mathbb{A}_1 \cap \mathbb{A}_2\}}{\mathbf{Pr}\{\mathbb{A}_2\}}. \tag{3.1}$$

For two independent events, $\mathbf{Pr}\{\mathbb{A}_1 | \mathbb{A}_2\} = \mathbf{Pr}\{\mathbb{A}_1\}$. The definition of conditional probability, expressed in Equation (3.1), can be generalised if all probabilities are conditioned to a third event $\mathbb{A}_3$: $\mathbf{Pr}\{\mathbb{A}_1 \cap \mathbb{A}_2 | \mathbb{A}_3\} = \mathbf{Pr}\{\mathbb{A}_2 | \mathbb{A}_3\} \cdot \mathbf{Pr}\{\mathbb{A}_1 | \mathbb{A}_2 \cap \mathbb{A}_3\}$. Indeed, by the definition of conditional probability, the right hand side can be written as:

$$\begin{aligned} \mathbf{Pr}\{\mathbb{A}_2 | \mathbb{A}_3\} \cdot \mathbf{Pr}\{\mathbb{A}_1 | \mathbb{A}_2 \cap \mathbb{A}_3\} &= \frac{\mathbf{Pr}\{\mathbb{A}_2 \cap \mathbb{A}_3\}}{\mathbf{Pr}\{\mathbb{A}_3\}} \cdot \frac{\mathbf{Pr}\{\mathbb{A}_1 \cap \mathbb{A}_2 \cap \mathbb{A}_3\}}{\mathbf{Pr}\{\mathbb{A}_2 \cap \mathbb{A}_3\}} \\ &= \frac{\mathbf{Pr}\{\mathbb{A}_1 \cap \mathbb{A}_2 \cap \mathbb{A}_3\}}{\mathbf{Pr}\{\mathbb{A}_3\}} \\ &= \mathbf{Pr}\{\mathbb{A}_1 \cap \mathbb{A}_2 | \mathbb{A}_3\}, \end{aligned} \tag{3.2}$$

where the last step descends from the very definition of conditional probability.

Given a random experiment and a sample space Ω, a random variable $\mathcal{X}$ is a function that reflects the result of a random experiment [87] associating the possible outcome of an experiment with a real value, $\mathcal{X} : \Omega \to \mathbb{R}$. For example, the number of jobs that arrive at a computer system or the time interval between the arrivals of two consecutive jobs at a computer system. Depending on the values that a random variable can assume it is possible to distinguish between continuous and discrete random variables.

A random variable $\mathcal{X}$ that can only assume discrete values is called a discrete random variable, where the discrete values are often non–negative integers. The set of the probabilities of each discrete values that the variable can assume is called the Probability Mass Function (PMF) of the random variable. Hence, the probability that the random variable $\mathcal{X}$ assumes the value x is defined as: $p_x = \mathbf{Pr}\{\mathcal{X} = x\}$.

On the other hand, a random variable $\mathcal{X}$ that can assume all values in the interval $[a, b]$, where $-\infty \leq a < b \leq +\infty$, is called a continuous random variable. It is described by its distribution function, also called Cumulative Distribution Function (CDF). Hence, the probability that the random variable $\mathcal{X}$ takes values less than or equal to k, for every k is defined as: $F_k = \mathbf{Pr}\{\mathcal{X} \leq k\}$.

A random process, also called stochastic process, is a collection of random variables $\{\mathcal{X}_t : t \in \mathcal{T}\}$, where each random variable $\mathcal{X}_t$ is indexed by the time parameter $t \in \mathcal{T}$. If a countable, discrete–time set $\mathcal{T}$ is encountered, the stochastic process is called a discrete–time process, $\mathcal{T}$ is commonly represented by a subset of $\mathbb{N}$ and the stochastic process is often referred to as a *chain*. A stochastic process models the evolution of a system over time.

3.2 Discrete–Time Markov Chains

A *Discrete–Time Markov Process* (DTMP) $\{\mathcal{X}_k\}$ is a discrete–time stochastic process such that its future development only depends on the current state and not on the past

history. This definition can be stated in formal terms on the conditional PMF:

$$\mathbf{Pr}\{\mathcal{X}_k = x_k \,|\mathcal{X}_1 = x_1,\, \mathcal{X}_2 = x_2,\, \ldots,\, \mathcal{X}_{k-1} = x_{k-1}\} = \mathbf{Pr}\{\mathcal{X}_k = x_k \,|\mathcal{X}_{k-1} = x_{k-1}\}\,.$$

A DTMP defined over a discrete state space is called Discrete–Time Markov chain (DTMC). Given a DTMC, let $\pi_h(k)$ represent the probability $\mathbf{Pr}\{\mathcal{X}_k = h\}$, π_k be the infinite vector $\pi_k = [\pi_0(k),\, \pi_1(k),\, \ldots]$, and $P = [p_{i,\,j}]$ be a matrix whose generic element $p_{i,\,j}$ is given by the conditional probability $p_{i,\,j} = \mathbf{Pr}\{\mathcal{X}_k = j \,|\mathcal{X}_{k-1} = i\}$, which can be interpreted as the probability to reach, in one step, the state j given that the system is currently in state i.

Consider a state i of a DTMC. This state i is called *transient* if there is some probability that, starting from i, the system will never return to i. The state i is *recurrent* if it is not transient. The *period* of a recurrent state i is defined as the greatest common divisor of the set of all numbers, n, for which $\mathbf{Pr}\{\mathcal{X}_m = i \wedge \mathcal{X}_{m+n} = i\} > 0$, $\forall m$. A state is called *aperiodic* if its period is equal to 1. A DTMC is called aperiodic, if all of its states are aperiodic.

The state i of a DTMC is called *positive recurrent* if its mean recurrence time is finite, and the DTMC is positive recurrent if all its states are positive recurrent. A DTMC is called *irreducible*, if every state can be reached from any other state in a finite number of steps. It can be shown that in an irreducible DTMC all states are of the same type. So, if one state is aperiodic, so is the DTMC.

Starting from an initial probability distribution π_0, the application of the Bayes theorem and of the properties of the Markov processes allow us to express the evolution of the distribution by the matrix equation $\pi_{k+1} = \pi_k \cdot P$. The matrix P is called *probability transition matrix*.

A very important property of irreducible and positive recurrent DTMC is *the existence of a single equilibrium*: $\tilde{\pi} = \tilde{\pi} \cdot P$, where the limiting distributions $\lim_{n\to\infty} \pi_n$ converge starting from any initial probability distribution π_0. This equilibrium is called *steady state distribution*.

A DTMC is called a Quasi–Birth–Death Process (QBDP) if its probability transition matrix P has the following block structure:

$$P = \begin{bmatrix} C & A_2 & \mathbf{0} & \mathbf{0} & \ldots \\ A_0 & A_1 & A_2 & \mathbf{0} & \ldots \\ \mathbf{0} & A_0 & A_1 & A_2 & \ddots \\ \mathbf{0} & \mathbf{0} & A_0 & A_1 & \ddots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{3.3}$$

where C, A_0, A_1, and A_2 are matrices and $\mathbf{0}$ denotes a block matrix of zero. When the matrices are scalars, this structure reduces to the standard Birth–Death Process (BDP).

3.3 A CPU Reservation as a Markov Chain

When the probability distribution of the inter–arrival and computation time are known independent identically distributed (i.i.d.) stochastic processes, the temporal isolation property allow us to model the evolution of a task scheduled through a resource reservation as a Discrete–time Markov chain with an infinite number of states [3, 4]. In this case, it has been proved that the DTMC describing the system takes the form of a QBDP [84].

In this setting, we will carry out a conservative analysis for each task assuming that it only receives its minimum guaranteed bandwidth with no interference from the other tasks. The result will be a lower bound for the probability of meeting the deadline.

For simplicity, it is assumed that the server period T^s of the reservation is constrained to be a sub–multiple of the release time of each job (the task period T). Let z_j be the integer multiple of T^s defining the distance between a_j and a_{j-1}: $a_j - a_{j-1} = z_j \cdot T^s$. Other choices are possible but make little practical sense.

Moreover, in the case of aperiodic tasks, the empirical probability distribution of the inter–arrival time used in the proposed model can be conservatively resampled, as presented in Section 3.4. In this case, the resample simulates that the jobs arrived before they actually did, accumulating the probabilities to the immediate inferior server period. By performing the stochastic analysis with a conservative approximation of the real distribution we introduce a certain level of pessimism in the resulting probabilistic guarantees, meaning that our analysis will estimate a probability of respecting the deadline which is smaller than the one provided by the system.

3.3.1 Dynamic Model

Let d_j^s denote the latest scheduling deadline used for job J_j and introduce the symbol $\delta_j = d_j^s - a_j$. The latest scheduling deadline d_j^s is an upper bound for the finishing time of the job. If Equation (2.3) is respected, then $f_j \leq d_j^s$, meaning that a job always finishes before its latest deadline. Hence, δ_j is an upper bound for the job response time.

Example 1 *Consider the schedule in Figure 3.1. The presented schedule considers a task τ_1 with two adjacent jobs starting at $a_{1,1}$ and $a_{1,2}$. The reservation period is chosen as one third of the task period. The job $J_{1,1}$, in this case, finishes beyond the deadline, which in our periodic model is $a_{1,2}$. More precisely, the last reservation period that the job $J_{1,1}$ uses, in which its finishing time lies, is upper–limited by the scheduling deadline d_1^s.*

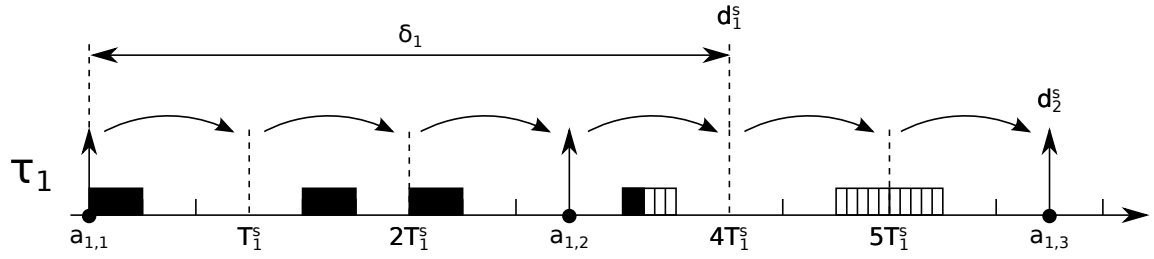

Figure 3.1: Example of a periodic real–time task τ_1 scheduled by a Constant Bandwidth Server. The two colours denote different jobs. At the activation of the first job (black rectangle), its initial scheduling deadline is set to $d_1^s = T_1^s$. Every time that the budget is depleted, the scheduling deadline is postponed by T_1^s (represented by the arrows). The latest scheduling deadline for the job ($4 \cdot T_1^s$) is an upper bound for the finishing time; hence, δ_j is an upper bound for the response time of the job.

The quantity δ_j takes on values in a discrete set: the integer multiples of T^s, and the probability $\eta = \mathbf{Pr}\{f_j \leq d_j\}$ of meeting a deadline $d_j = a_j + D$ is lower bounded by $\mathbf{Pr}\{\delta_j \leq D\}$: $\eta \geq \mathbf{Pr}\{\delta_j \leq D\}$.

It is possible to express the dynamic evolution of δ_j using the following stochastic model [4]:

$$\begin{aligned} v_1 &= c_1 \\ v_j &= \max\{0, v_{j-1} - z_j \cdot Q^s\} + c_j \\ \delta_j &= \left\lceil \frac{v_j}{Q^s} \right\rceil \cdot T^s \end{aligned} \tag{3.4}$$

In the following, we will use the function $[a]^+ = \max\{0, a\}$. It is useful to observe that the workload v_j, representing the amount of backlogged computation time that has to be served by the CBS scheduler when a new job arrives, is not directly measurable. It can only be evaluated at the *end* of the job through the measurement of δ_j.

Since the process modelling the sequence c_j of the computation time is assumed a discrete valued and i.i.d. stochastic process, the model in Equation (3.4) represents a DTMC, where the states are determined by the possible values of v_j and the transition probabilities, by the PMF of the computation time $\mathcal{C}(c) = \mathbf{Pr}\{c_j = c\}$. Figure 3.2 shows a graphical representation of a DTMC describing a resource reservation where the computation time of the task is described by an independent and identically distribution stochastic process.

Let us introduce the symbols $c^{\min}$ and $c^{\max}$, to denote the minimum and maximum com-

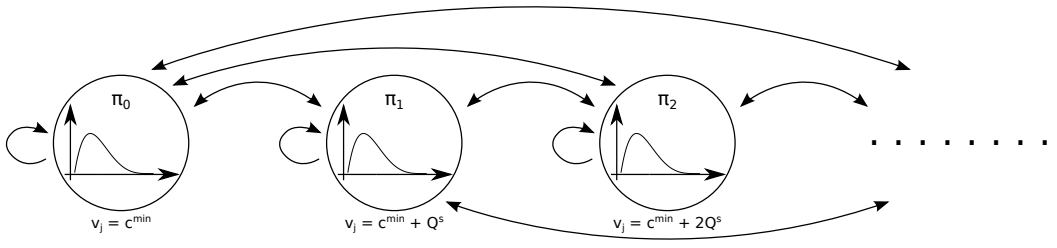

Figure 3.2: Graphical representation of the dynamic evolution of the states of the system, modelled as an infinite queue. The states are given by the possible values (infinite) of the workload v_j. The probability of such values is given by $\pi_h(j) = \mathbf{Pr}\left\{v_j = c^{\min} + h\right\}$ and the evolution of the vector $\Pi = [\pi_0, \pi_1, \ldots]$ can be described using the standard notation of the Markov chains.

putation time. Since the computation time varies in the bounded set $\left\{c^{\min}, \ldots, c^{\max}\right\}$, the value of v_j will be lower bounded by $c^{\min}$. Let us define the event $\mathcal{V}_h(j) = \left\{v_j = c^{\min} + h\right\}$, meaning that the workload is equal to $c^{\min} + h$ at job j, with $h \in [0, \ldots, \infty[$ representing the total backlog accumulated until the previous job (v_{j-1}).

The generic element of the probability transition matrix P can be computed observing that the evolution of the system is given by Equation (3.4). In particular, the element $P^{(h,h')}$, with row h and column h', represents the probability of transition from $\mathcal{V}_h(j-1)$ to $\mathcal{V}_{h'}(j)$, which is given by:

$$
\begin{aligned}
P^{(h,h')} &= \mathbf{Pr}\left\{\mathcal{V}_{h'}(j) \,|\mathcal{V}_h(j-1)\right\} \\
&= \mathbf{Pr}\left\{v_j = c^{\min} + h' \,\middle|\, v_{j-1} = c^{\min} + h\right\} \\
&= \mathbf{Pr}\left\{\left[v_{j-1} - z_j \cdot Q^s\right]^+ + c_j = c^{\min} + h' \,\middle|\, v_{j-1} = c^{\min} + h\right\} \\
&= \mathbf{Pr}\left\{\left[c^{\min} + h - z_j \cdot Q^s\right]^+ + c_j = c^{\min} + h'\right\} \\
&= \mathbf{Pr}\left\{c_j = c^{\min} + h' - \left[c^{\min} + h - z_j \cdot Q^s\right]^+\right\} \\
&= \sum_{z=0}^{\infty} \mathbf{Pr}\left\{z_j = z\right\} \cdot \mathbf{Pr}\left\{c_j = c^{\min} + h' - \left[c^{\min} + h - z \cdot Q^s\right]^+\right\}.
\end{aligned}
\tag{3.5}
$$

Assume that the inter–arrival time z_j is defined in the bounded set $\left\{z^{\min}, \ldots, z^{\max}\right\}$.

Let us introduce the following definitions:

$$
\begin{aligned}
a_{h,\,h'} &= \mathbf{Pr}\left\{c_j = c^{\min} + h'\right\} \\
b_{h,\,h'} &= \sum_{z=z^{\min}}^{z^{\max}} \mathbf{Pr}\left\{z_j = z\right\} \cdot \mathbf{Pr}\left\{c_j = c^{\min} + h' - \left[c^{\min} + h - z \cdot Q^s\right]^+\right\} \\
D &= c^{\max} - c^{\min} \\
E &= (z^{\min} \cdot Q^s) - c^{\min} + 1 \\
F &= (z^{\max} \cdot Q^s) - c^{\min} + 1 \\
G &= c^{\max} - c^{\min} + ((z^{\max} - z^{\min}) \cdot Q^s)
\end{aligned}
$$

Hence, the probability transition matrix P developed from Equation (3.5) can be written as:

$$
P = \begin{bmatrix}
a_{0,\,0} & a_{0,\,1} & \dots & a_{0,\,D} & 0 & 0 & \dots & \dots & \dots & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a_{E,\,0} & a_{E,\,1} & \dots & a_{E,\,D} & 0 & 0 & \dots & \dots & \dots & \dots \\
b_{E+1,\,0} & b_{E+1,\,1} & \dots & b_{E+1,\,D} & b_{E+1,\,D+1} & 0 & 0 & \dots & \dots & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
b_{F,\,0} & b_{F,\,1} & \dots & b_{F,\,D} & b_{F,\,D+1} & \dots & b_{F,\,G} & 0 & 0 & \dots \\
0 & b_{F,\,0} & b_{F,\,1} & \dots & b_{F,\,D} & b_{F,\,D+1} & \dots & b_{F,\,G} & 0 & 0 \\
0 & 0 & b_{F,\,0} & b_{F,\,1} & \dots & b_{F,\,D} & b_{F,\,D+1} & \dots & b_{F,\,G} & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{bmatrix}, \tag{3.6}
$$

which has the recursive structure shown in Equation (3.3). Indeed, from row F onward, each row is obtained by shifting the previous one to the right and inserting a 0 in the first position.

Let us define $\pi_h(j) = \mathbf{Pr}\left\{\mathcal{V}_h(j)\right\} = \mathbf{Pr}\left\{v_j = c^{\min} + h\right\}$. In plain words, $\pi_h(j)$ represents the probability that at the j^{th} job, the workload v_j be $c^{\min} + h$. Introduce the vector $\Pi(j)$, namely the vector of probabilities associated with the state $\mathcal{V}(j)$:

$$
\Pi(j) = [\pi_0(j),\, \pi_1(j),\, \ldots]. \tag{3.7}
$$

In general, we cannot rule out the possibility that the state reaches arbitrarily large values (although with a small probability). Thereby, the probability vector has an infinite size. The evolution of the vector $\Pi(j)$, presented in Equation (3.7), can be described using the standard notation of the Markov chains [26]: $\Pi(j) = \Pi(j-1) \cdot P$.

3.3.2 Computation of the Steady State Probability

The probability transition matrix P describes the evolution of the probability of finding the system in each state starting from an initial distribution. In our case, the vector $\Pi(j)$,

presented in Equation (3.7), is made of an infinite number of elements, and so will be the matrix P.

The four matrices C, A_0, A_1, A_2 describing the QBDP, as presented in Equation (3.3), can be expressed in terms of the probability transition matrix P in Equation (3.6). In order to see this, let us introduce the symbol $P(p_z : p_Z\,;\, q_z : q_Z)$ to denote the sub–matrix obtained from P extracting the block of rows from p_z to p_Z and the block of columns from q_z to q_Z. By setting $H = \max\{D+1,\, F+1\}$, we have:

$$\begin{aligned} C &= P(1 : H\,;\, 1 : H), \\ A_2 &= P(1 : H\,;\, H+1 : 2H), \\ A_0 &= P(H+1 : 2H\,;\, 1 : H), \\ A_1 &= P(H+1 : 2H\,;\, H+1 : 2H). \end{aligned} \tag{3.8}$$

Through easy, but tedious, computations it is possible to show that by setting the matrices as in Equation (3.8), the transition matrix indeed reduces to the QBDP structure mentioned above. After casting our system into the QBDP framework, we can capitalise on the rich body of results in the field. In particular, it is possible to show that:

1. The system admits a steady state

$$\begin{aligned} \widetilde{\Pi} &= \lim_{j\to\infty} \Pi(j) \\ &= \lim_{j\to\infty} [\pi_0(j),\, \pi_1(j),\, \ldots]. \end{aligned} \tag{3.9}$$

2. The steady state distribution $\widetilde{\Pi}$ is unique and independent from the initial distribution $\Pi(0)$,

3. The computation of the steady state probability can be done using the very efficient numeric solutions available in the literature. In particular, this thesis uses the Cyclic Reduction algorithm presented in [18] for the numeric solution, although the Logarithmic Reduction algorithm [66] is also suitable for this purpose.

3.3.3 Computation of the Distribution of the Response Time

From the steady state probability, it is possible to recover the steady state distribution of the variable δ_j, which, as mentioned before, is an upper bound of the task's response time. Considering that $\mathbf{Pr}\{\lceil a \rceil = b\} = \mathbf{Pr}\{b-1 < a \leq b\}$, the steady state CDF can be

reconstructed using the following formula:

$$\begin{aligned}\lim_{j\to\infty}\mathbf{Pr}\left\{\delta_j = \delta\cdot T^s\right\} &= \lim_{j\to\infty}\mathbf{Pr}\left\{\left\lceil\frac{v_j}{Q^s}\right\rceil = \delta\right\}\\ &= \lim_{j\to\infty}\mathbf{Pr}\left\{\delta-1<\frac{v_j}{Q^s}\leq\delta\right\}\\ &= \lim_{j\to\infty}\mathbf{Pr}\left\{(\delta-1)\cdot Q^s < v_j \leq \delta\cdot Q^s\right\}\\ &= \sum_{h=((\delta-1)\cdot Q^s)+1}^{\delta\cdot Q^s}\lim_{j\to\infty}\mathbf{Pr}\left\{v_j = h\right\},\end{aligned} \tag{3.10}$$

where:

$$\lim_{j\to\infty}\mathbf{Pr}\left\{v_j = h\right\} = \begin{cases}0 & \text{if } h < c^{\min}\\ \lim_{j\to\infty}\mathbf{Pr}\left\{\mathcal{V}_h(j)\right\} & \text{otherwise}\end{cases}.$$

3.4 A Conservative Approximation

In order to make the model tractable from the numeric point of view, it is useful to introduce a conservative approximation. The notion of conservative approximation adopted in this thesis relies on the concept of *first order stochastic dominance*.

Definition 1 *Given two random variables $\mathcal{X}$ and $\mathcal{Y}$, with CDFs $F_x = \mathbf{Pr}\{\mathcal{X}\leq x\}$ and $F_y = \mathbf{Pr}\{\mathcal{Y}\leq y\}$, $\mathcal{X}$ has a first order stochastic dominance over $\mathcal{Y}$, represented as $\mathcal{X}\succeq\mathcal{Y}$, if and only if $\forall x\ F_x \leq F_y$.*

Graphically, this means that the curve F_x never goes above the curve F_y. Note that if the curves F_x and F_y cross, the variables $\mathcal{X}$ and $\mathcal{Y}$ are not comparable, and it is not true that $\mathcal{X}\succeq\mathcal{Y}$ nor $\mathcal{Y}\succeq\mathcal{X}$. Figure 3.3 shows an example of this concept where $\mathcal{Y}$ and $\mathcal{Z}$ are a conservative approximation of $\mathcal{X}$ given that $\mathcal{Y}\succeq\mathcal{X}$ and $\mathcal{Z}\succeq\mathcal{X}$. However, since $\mathcal{Y}$ and $\mathcal{Z}$ intersect each other, $\mathcal{Y}\not\succeq\mathcal{Z}$ and $\mathcal{Z}\not\succeq\mathcal{Y}$.

Based on this definition, a stochastic real–time task can be seen as a conservative approximation of another one if its probabilistic deadlines are stochastically dominated by the probabilistic deadlines of the original task: considering δ_j in Equation (3.4), this plainly means that in the modified system, the low values of the δ_j will have a greater probability and so will be the probability of the first element of the probability vector (associated with the deadline satisfaction).

As shown by Diaz et al. [39], if $\mathcal{C}'$ stochastically dominates $\mathcal{C}$, then a system having the computation times distributed according to $\mathcal{C}'$ is a conservative approximation of the original system with the computation times distributed according to $\mathcal{C}$.

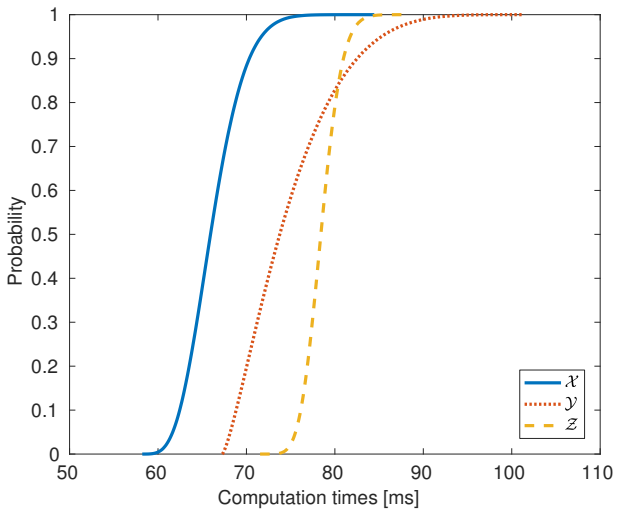

Figure 3.3: Graphical meaning of the first order stochastic dominance. Both $\mathcal{Y}$ and $\mathcal{Z}$ have first order stochastic dominance over $\mathcal{X}$, represented as $\mathcal{Y} \succeq \mathcal{X}$ and $\mathcal{Z} \succeq \mathcal{X}$, because their CDFs never go above the CDF of $\mathcal{X}$. In the case of $\mathcal{Y}$ and $\mathcal{Z}$, as they intersect there is no stochastic dominance between them: $\mathcal{Y} \not\succeq \mathcal{Z}$ and $\mathcal{Z} \not\succeq \mathcal{Y}$.

A simple way to build $\mathcal{C}'$ to obtain such a conservative approximation is to replace c_j with a new variable c'_j, taking values in the interval $[c^{\min}, c^{\max}]$ and whose distribution is given by:

$$\mathcal{C}'_{\Delta}(c') = \begin{cases} 0 & \text{if } (c' \bmod \Delta) \neq 0 \\ \sum_{c=((k-1)\cdot\Delta)+1}^{k\cdot\Delta} \mathcal{C}(c) & \text{otherwise} \end{cases}, \tag{3.11}$$

where k takes values in the interval $\left[1, \left\lceil \frac{c^{\max}}{\Delta} \right\rceil\right]$ and Δ is called the `granularity`, which is a scaling factor chosen as an integer sub–multiple of Q^s used to resample the distribution.

Figure 3.4 shows an example of the application of the conservative approximation technique using a scaling factor $\Delta = 2$. Note that the resulting PMF has zeros in the odd indexes while the even indexes (multiples of Δ) accumulate the probabilities of the indexes smaller than or equal to the current one (up to the previous multiple of Δ). For instance, $\mathcal{C}(2) = \mathcal{C}'_2(2) = 0.02$ because there are no probabilities to accumulate ($\mathcal{C}(1) = 0.0$). On the other hand, $\mathcal{C}'_2(4) = 0.17$ accumulates the probabilities of the indexes smaller than or equal to 4 ($\mathcal{C}(3) = 0.07$ and $\mathcal{C}(4) = 0.1$). It is also important to note that the choice of a PMF or a PDF to describe the computation times does not affect the applicability of the conservative approximation, which can be regarded as a *quantisation* of the original distribution.

This resampled distribution of the computation times, $\mathcal{C}'$, generates a new DTMC, whose transition matrix has again the recursive structure shown in Equation (3.3), where

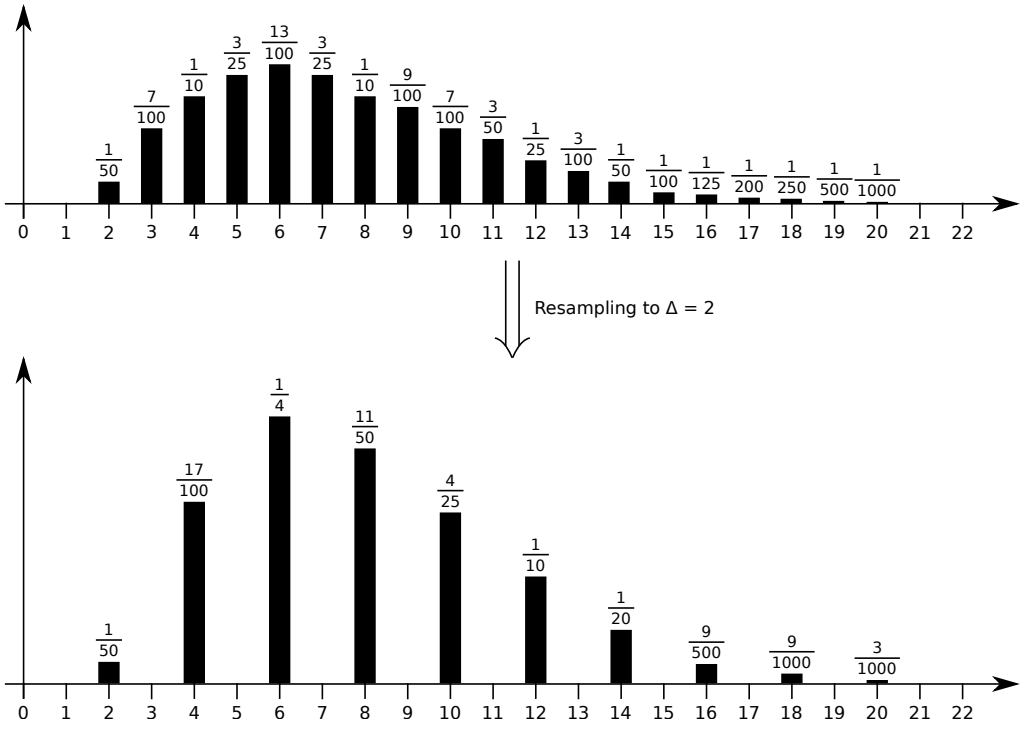

Figure 3.4: Example of the application of the conservative approximation. The original distribution (top) is resampled using a scaling factor $\Delta = 2$. The resampled distribution (bottom) is obtained by accumulating the probabilities from odd indexes together with the probabilities of the even indexes placed immediately to the right. For instance, $\mathcal{C}'_2(4) = \mathcal{C}(3) + \mathcal{C}(4) = 0.07 + 0.1 = 0.17$.

the different elements of the matrix are functions of the parameter Δ. Large values of Δ correspond to a smaller size for matrices C, A_0, A_1 and A_2 in Equation (3.3). This reduces the time required for the computation of the steady state probability paying the price of a coarser approximation for the computed probability.

A similar approach can be followed to obtain a conservative approximation of the probability distribution of the inter–arrival times. In this case, the scaling factor is set to $\Delta = T^s$ and the accumulation of the probabilities is performed towards the lower index, meaning that all the probabilities of activations occurred between two reservations periods, for instance in the range $[T^s, 2 \cdot T^s)$, are accumulated at time T^s. Hence, the resample process models the activations as if they occurred before they actually did. This strategy allows us to represent the probability of inter–arrival time with a model more conservative than the original distribution.

Chapter 4

I.I.D. Computation Times[1]

This chapter presents a methodology for the computation of the probability of deadline miss for a periodic real–time task scheduled by a resource reservation algorithm. The proposed techniques take advantage of the periodic structure characterising a Quasi–Birth–Death–Process (QBDP) whose transition matrix is reported in Equation (3.6).

This structure is exploited to develop an efficient numeric solution where different accuracy/computation time trade–offs can be obtained by operating on the granularity (scaling factor Δ) of the model. Specifically, we have assumed the conservative approximation discussed in Section 3.4.

More importantly we have developed an analytic solution, namely a closed form conservative bound, for the probability of a deadline miss. Our experiments reveal that the bound remains reasonably close to the experimental probability obtained from our test case real–time applications.

4.1 A Numeric Algorithm

The first key result shows a general expression for the steady state probability of respecting the deadline. As mentioned before, the analysis is restricted to periodic tasks, hence the

[1]The theoretical results of this chapter are mainly reported for completeness since they were developed before the PhD studies of the author, who actively participated in the experimental validation of those results.

transition matrix presented in Equation (3.6) reduces to the following matrix:

$$P = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,D} & 0 & 0 & \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{E,0} & a_{E,1} & \dots & a_{E,D} & 0 & 0 & \dots & \dots & \dots & \dots \\ 0 & a_{E,0} & a_{E,1} & \dots & a_{E,D} & 0 & 0 & \dots & \dots & \dots \\ 0 & 0 & a_{E,0} & a_{E,1} & \dots & a_{E,D} & 0 & 0 & \dots & \dots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{4.1}$$

which retains the recursive structure shown in Equation (3.3). Indeed, from row E onward, each row is obtained by shifting the previous one to the right and inserting a 0 in the first position. For completeness, recall the following definitions:

$$\begin{aligned} a_{h,h'} &= \mathbf{Pr}\left\{c_j = c^{\min} + h'\right\} \\ D &= c^{\max} - c^{\min} \\ E &= (z^{\min} \cdot Q^s) - c^{\min} + 1 \end{aligned}$$

Let us define the following function $\varphi : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ as:

$$\varphi_{k,l} = \sum_{j=0}^{k} \alpha_j \cdot l^{k-j},$$

where $\alpha_j = a_{0,j}/a_{0,0}$. Using this function and the structure of the QBDP, it is possible to write the equation expressing the steady state equilibrium $\Pi(j) = \Pi(j) \cdot P$, where $\Pi(j) = [\pi_0(j), \pi_1(j), \ldots]$, as defined in Equation (3.7). The steady state solution is given by the following theorem:

Theorem 1 *Consider a QBDP described by the transition probability matrix P given in Equation* (4.1)*, in which both $a_{0,0}$ and $a_{0,D}$ differ from zero. Assume that the matrix:*

$$W = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 1 \\ -\alpha_D & -\alpha_{D-1} & -\alpha_{D-2} & \dots & w & -\alpha_{H-1} & \dots & -\alpha_1 \end{bmatrix} \tag{4.2}$$

where $H = E - 1$ and $w = \varphi_{H-1,1} + \sum_{j=H+1}^{D} \alpha_j$, has distinct eigenvalues. Let $\tilde{\pi}_j = \lim_{k \to +\infty} \pi_j(k)$ be the steady state distribution of the state. One of the two following cases apply:

(I) If $\sum_{j=0}^{H-1} \varphi_{j,1} \leq \sum_{j=H+1}^{D} (j-H) \cdot \alpha_j$ *then the limiting distribution is given by:*

$$\tilde{\pi}_j = \lim_{k \to +\infty} \tilde{\pi}_j(k) = 0, \forall j, \tag{4.3}$$

(II) If $\sum_{j=0}^{H-1} \varphi j,\ 1 > \sum_{j=H+1}^{D} (j-H) \cdot \alpha_j$ *then:*

$$\tilde{\pi}_0 = \prod_{\beta \in \mathcal{B}_s} (1-\beta). \tag{4.4}$$

Proof. The proof of this theorem is presented in [84]. □

In the second case, $\mathcal{B}_s$ is the set of stable eigenvalues of W (in this context an eigenvalue β is said stable if $|\beta| < 1$), and the terms $\tilde{\pi}_j$ with $0 < j < H$ are known linear functions of $\tilde{\pi}_0$, while the terms $\tilde{\pi}_j$ with $j \geq H$ are given by:

$$\begin{aligned} \tilde{\pi}_H &= \sum_{j=H+1}^{D} \alpha_j \cdot \tilde{\pi}_0 - \sum_{j=1}^{H-1} \varphi_{j,1} \cdot \tilde{\pi}_{H-j}, \\ \tilde{\pi}_{H+l} &= \left(\varphi_{H-1,1} + \sum_{j=H+1}^{D} \alpha_j \right) \cdot \tilde{\pi}_l - \sum_{\substack{j=1 \\ j \neq H}}^{min(D, l+H)} \alpha_j \cdot \tilde{\pi}_{l+H-j}, \end{aligned} \tag{4.5}$$

holding $\forall l \geq 1$.

At this point, it is important to make two important remarks:

Remark 1 *The assumption on the eigenvalues of the matrix* W *is merely technical and it is not restrictive. In all our examples (both synthetically generated and using data from real applications), it is respected. We believe that artificial examples that violate it could probably be constructed but they are not relevant in practice.*

Remark 2 *As well as paving the way for the analytical bound presented in Theorem 2, Theorem 1 contains an implicit numeric algorithm for the computation of the probability of respecting a deadline equal to the task period (*$\tilde{\pi}_0$*) based on the computation of the eigenvalues of the matrix* W*. Since the latter is in companion form, in the following we refer to this algorithm as* `companion`*.*

The rationale behind Theorem 1 is the following. First, the equilibrium point of the QBDP is expressed as an iterative system. The evolution in the iteration step represents the connection between the probabilities of the different states. Using this representation and some property of convergence of the Markov chain, we can express all the steady–state probabilities as a function of $\tilde{\pi}_0$, which can eventually be found as a solution of a linear system of equations.

The equilibrium of the QBDP can be derived, exploiting Equation (4.2) and Equation (4.5), by the following iterative equation for the vector $\Pi_j = [\tilde{\pi}_j, \ldots, \tilde{\pi}_{j+D-1}]^T$:

$$\Pi_1 = \begin{bmatrix} \tilde{\pi}_1 \\ \tilde{\pi}_2 \\ \vdots \\ \tilde{\pi}_D \end{bmatrix} = W \cdot \Pi_0 \Rightarrow \Pi_j = \begin{bmatrix} \tilde{\pi}_j \\ \tilde{\pi}_{j+1} \\ \vdots \\ \tilde{\pi}_{D-1+j} \end{bmatrix} = W^j \cdot \Pi_0. \tag{4.6}$$

The characteristic polynomial of the *companion form* matrix W reported in Equation (4.2) is given by:

$$P(\lambda) = \lambda^n - \left(\gamma_{H-1,1} + \sum_{j=H+1}^{n} \alpha_j \right) \lambda^{n-H} + \sum_{\substack{j=1 \\ j \neq H}}^{n} \alpha_j \lambda^{n-j}, \tag{4.7}$$

where the terms α_i of the polynomial allow us to obtain the steady–state probabilities for the next step of the iteration based on the steady–state probabilities of the previous iteration, as presented in Equation (4.6).

4.2 An Analytical Bound

As discussed earlier, the steady state probability of meeting a deadline equal to the task period can be found by computing the first element $\tilde{\pi}_0$ of the $\widetilde{\Pi}$ that solves the equation $\widetilde{\Pi} = \widetilde{\Pi} \cdot P$, where P is the infinite transition matrix in Equation (4.1) associated with the Discrete–time Markov chain (DTMC). Let us consider a new DTMC whose transition matrix is given by:

$$P' = \begin{bmatrix} b'_H & a_{0,H+1} & \ldots & a_{0,D} & 0 & 0 & \ldots \\ a'_{H-1} & a_{0,H} & a_{0,H+1} & \ldots & a_{0,D} & 0 & \ldots \\ 0 & a'_{H-1} & a_{0,H} & a_{0,H+1} & \ldots & a_{0,D} & 0 \\ 0 & 0 & a'_{H-1} & a_{0,H} & a_{0,H+1} & \ldots & a_{0,D} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{4.8}$$

where $H = E - 1$, $b'_H = \sum_{k=0}^{H} a_{0,k}$ and $a'_{H-1} = \sum_{k=0}^{H-1} a_{0,k}$.

The underlying idea is very simple. Consider the DTMC associated with matrix P. The terms on the left of the diagonal are transition probabilities toward states with a smaller delay than the current one. By using P', we lump together all these transitions to the state immediately on the left of the current one. For instance, if the current state corresponds to 4 server periods of delay, its only enabled transition to the left will be

to the state associated with delay 3. The effect of deleting the transition toward states associated with smaller delays is to slow down the convergence toward small delays, thus decreasing the steady state probability of these states.

Let $\widetilde{\Pi}'$ represent the steady state probability of this system. We can easily show the following:

Lemma 1 *Let Γ be a random variable representing the state of the DTMC evolving with transition matrix P and Γ' be a random variable describing the state of the DTMC associated with the transition matrix P'. If both DTMC are irreducible and aperiodic, then, at the steady state, Γ' has a first order stochastic dominance over Γ: $\Gamma' \succeq \Gamma$, according to Definition 1. Therefore, for the first element of the steady state probability, we have $\tilde{\pi}_0 \geq \tilde{\pi}'_0$.*

Proof. The proof of this lemma is presented in [5]. □

In view of Lemma 1, we can focus on the system associated to the transition matrix P'. In this case, we immediately derive that the equilibrium condition $\widetilde{\Pi}' = \widetilde{\Pi}' \cdot P'$ produces the following recursion:

$$\begin{aligned} \tilde{\pi}'_1 &= \sum_{k=2}^{D} \alpha_k \cdot \tilde{\pi}'_0, \\ \tilde{\pi}'_l &= \left(1 + \sum_{k=2}^{D} \alpha_k\right) \cdot \tilde{\pi}'_{l-1} - \sum_{k=2}^{\min(D,\, H+l-1)} \alpha_k \cdot \tilde{\pi}'_{l-j}, \end{aligned} \tag{4.9}$$

holding $\forall l > 1$. These equations, as well as P', have been respectively derived from Equation (4.5) and P by imposing $H = 1$. In this situation, the following theorem holds:

Theorem 2 *Consider a QBDP described by the transition probability matrix presented in Equation* (4.8)*, in which both $a_{0,\,D}$ and a'_{H-1} differ from zero. Assume that the matrix W in Equation* (4.2) *has distinct eigenvalues after imposing $H = 1$. Then, there exists a limiting probability distribution given by:*

$$\begin{aligned} \tilde{\pi}'_0 &= \lim_{k \to +\infty} \pi'_0(k) \\ &= \max\left\{1 - \sum_{k=2}^{D} (k-1) \cdot \alpha_k,\, 0\right\} \\ &= \max\left\{1 - \sum_{k=2}^{D} (k-1) \cdot \frac{a_k}{a_0},\, 0\right\}, \end{aligned} \tag{4.10}$$

while the generic terms $\pi^{(j)}$, with $j > 0$, are given by (4.9).

Proof. The proof of this theorem is presented in [84]. □

The intuitive meaning of this result is the following. Consider a DTMC with transition matrix as in Equation (4.1) and assume for simplicity $D = 4$ and $H = 1$. The analytical bound in Theorem 2 is given by:

$$\begin{aligned}\tilde{\pi}'_0 &= 1 - 3 \cdot \alpha_4 - 2 \cdot \alpha_3 - \alpha_2 \\ &= 1 - 3 \cdot \frac{a_4}{a_0} - 2 \cdot \frac{a_3}{a_0} - \frac{a_2}{a_0}\end{aligned}$$

In the computation of the steady state probability $\tilde{\pi}_0$ we have to consider every possible transition to the right (i.e., increasing the delay) that the system can make. For each of them, we compute the ratio between the probability of taking the transition and the aggregate probability of moving to the left (decreasing the delay). In the final computation each of this ratio has a state proportional to the delay introduced. In our example, a_4 corresponds to three steps to the right and is weighted by the factor 3.

The application of this result to our context can be formalised in the following corollary:

Corollary 1 *Consider a resource reservation used to schedule a periodic task and suppose that the QBDP produced respects the assumption in Theorem 1. Then the probability of respecting the deadline is greater than or equal to:*

$$\tilde{\pi}'_0 = 1 - \sum_{k=2}^{D} (k-1) \cdot \frac{\mathcal{C}'_{\Delta}(z^{min} + k - 1) \cdot Q^s)}{\sum_{h=0}^{z^{min}-1} \mathcal{C}'_{\Delta}(h \cdot Q^s)} \tag{4.11}$$

This corollary descends from the following facts:

1. The DTMC described by the matrix P in Equation (4.1) is a conservative approximation of the system;
2. Lemma 1 provides an analytically tractable approximation of the DTMC with transition matrix P'; and,
3. Theorem 1 and Theorem 2 contain the analytical bounds.

4.3 Experimental Validation

We have validated the presented approach in two different ways, both using real applications. First, we have computed the probabilistic deadline using a 3D mapping and navigation application, to compare the accuracy and efficiency of the analytic bound against other methods and to assess the impact of the scaling factor Δ, presented in

Equation (3.11), and of the bandwidth. This set of experiments reveals a very good performance of the bound for appropriate choices of the scaling factor Δ. Its very low computation time allows one to select the best choice of Δ by testing a number of alternative choices. The tightness of the bound improves when the bandwidth is sufficient to achieve an acceptable real–time behaviour for the application.

In a second set of experiments, we have evaluated the method on a robotic control application, for which the mathematical assumptions underlying the model do not strictly apply. Hence, the produced results are obviously approximate; however the good quality of the approximation makes an interesting case for the practical applicability of the methodology.

4.3.1 The 3D Map Navigator Application

The first application consists of a 3D map navigator developed using WebGL. This application is part of the intelligent robotic walker, the "FriWalk", developed in the context of the European Project ACANTO[2]. The FriWalk is a robotic walking assistant supporting elderly users in their daily activities, and the test case application allows the user to navigate the environment.

The 3D map navigator application requires three pieces of information from the FriWalk to move a placeholder on the map: its latitude, longitude and orientation. After receiving this data, the navigator properly moves the placeholder on the map. This step involves several operations, for instance requesting a new tile for the map or creating new 3D objects. Consequently, the WebGL engine is responsible for drawing the objects on the screen.

In order to bypass the client–server architecture of X11, which is not suitable for real–time 3D graphics and rendering, the application relies on the solution proposed in [74] making suitable for real–time applications to use fixed–priority scheduling and CPU reservations.

The studied scenario consisted of several random walks on a given environment. The starting points of the navigation and the orientation of the camera are randomly selected at each run. The FriWalk moves at 1 m/s and the redrawing of the 3D map occurs five times per second (once every 200 ms). The systems records the computation time required to draw each frame. This computation time is obtained by using the Google Chrome's Trace Event Profiler tool.

All the results presented in this section have been measured on a Inspiron 15R 5521 equipped with an Intel Core i7–3537U 2–core processor operated at 2.0 Ghz and with

[2]A CyberphysicAl social NeTwOrk using robot friends. http://www.ict-acanto.eu/acanto

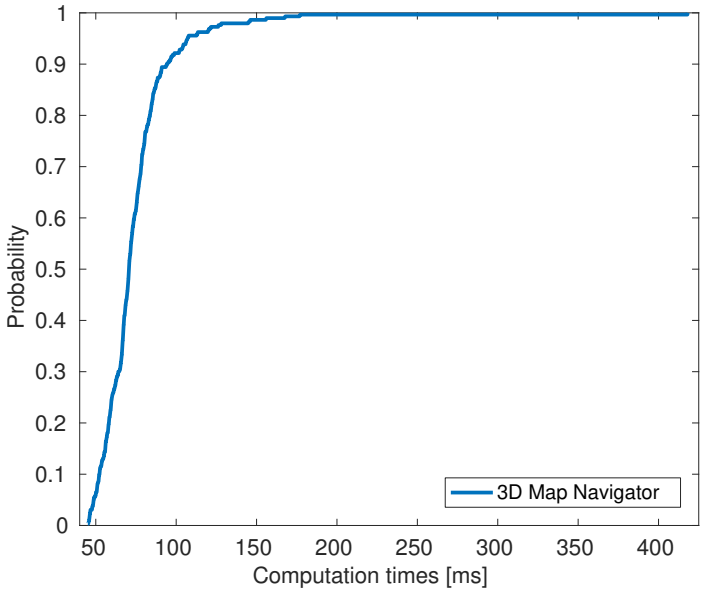

Figure 4.1: Empirical cumulative distribution function for the computation times experimentally obtained (measured) from the 3D Map Navigator application.

8 GB of RAM. The hyperthreading and the CPU power management were disabled for the experiments while the frequency switching governor was set to `performance`.

We report the results of the comparison between the numeric solution resulting from Theorem 1 and discussed in Remark 2 (`companion`), the analytic approximated bound presented in Corollary 1 (`analytic`) and the Cyclic Reduction algorithm [18] (`cyclic`). We have chosen `cyclic` after a selection process in which several algorithms for the solution of general QBDP problems and implemented in the SMCSolver tool–suite [20] were tested on a set of example QBDPs derived from our application. The different algorithms have been implemented in C++ in the PROSIT[3] [99] tool, described more in detail in Chapter 6.

As mentioned before, we report below the results obtained for a periodic task with period $T = 200$ ms and computation time described by an i.i.d stochastic process. Figure 4.1 shows the empirical cumulative distribution function of the computation times.

Effect of Δ

A first set of experiments was conducted to evaluate the impact of the Δ scaling factor. We considered as scheduling parameters: reservation period $T^s = 50$ ms and budget $Q^s = 24$ ms. Figure 4.2 shows the results for the probability of respecting a deadline

[3]https://bitbucket.org/luigipalopoli/prositool.git

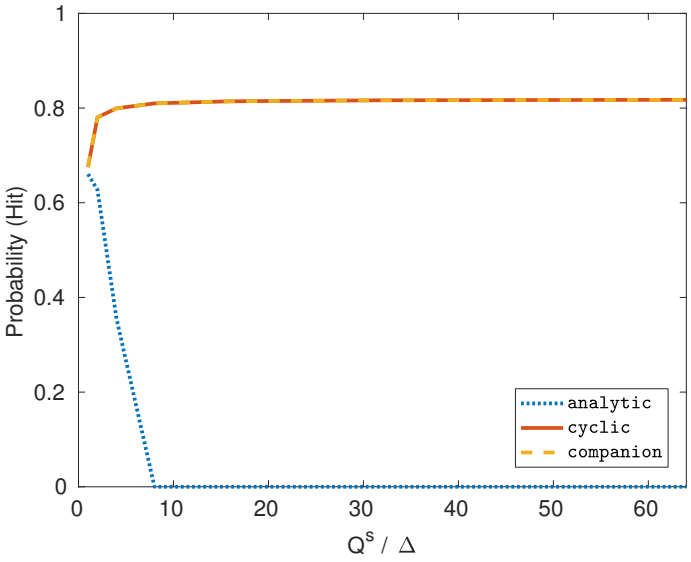

Figure 4.2: Impact of the scaling factor Δ on the accuracy of the computed probability of respecting a deadline equal to the task period. In the case of the `cyclic` and `companion` methods, as the scaling factor is reduced, the accuracy in the probability of deadline hit improves. On the other hand, for the `analytic` method, the scaling factor must be kept large to obtain accurate results.

equal to the task period achieved for different values of Δ (chosen as sub–multiples of Q^s).

In accordance with our expectations, `cyclic` and `companion` produce almost the same result in terms of probability (differences are from the 8^{th} digit) and the probability changes monotonically with Δ. For example: for $\Delta = Q^s$, the coarsest possible granularity, the value of the probability is 0.68, while it is 0.82 for $\Delta = Q^s/64$. The reason for this decrease is obvious since resampling introduces a conservative approximation and the error is larger for increasing granularity.

For the `analytic` bound the computed probability is not monotonic with Δ. In our example, the probability decreases from 0.66 at $\Delta = Q^s$ to 0.36 at $\Delta = Q^s/4$, and then becomes 0.0001 at $\Delta = Q^s/8$. The reason is that in the `analytic` bound we have two distinct effects, which play in opposite directions. On one hand, if we reduce Q^s we have the same conservative approximation effect as for `cyclic` or for any other numeric method. On the other hand, as explained before, lumping together all backward transitions reduce the recovery of the error when the computation demand is smaller than the allocated bandwidth.

Another factor that we analysed was the computation time required by PROSIT to compute the probabilistic guarantees. In this case, each analysis was performed 50 times.

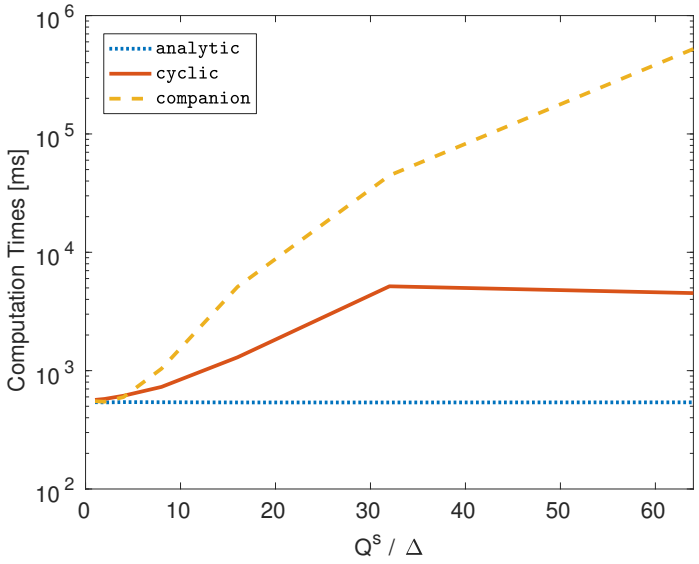

Figure 4.3: Impact of the scaling factor Δ on the computation time required by PROSIT to perform the stochastic analysis. The time required by the `analytic` method is not affected by the variations in Δ. The computation time required by the `cyclic` method suffers (up to a certain level) the changes of Δ. Finally, the `companion` method is highly sensitive to the variations of Δ, as the time exponentially grows by reducing Δ.

Hence, the computation times reported here correspond to the average time of the individual trials with its corresponding 95% confidence interval. These results are presented in Figure 4.3. For both `cyclic` and `companion`, the computation time changes with Δ in a substantial way. For instance: for `cyclic` the computation time is 566.06 $\pm$ 9.2 ms at $\Delta = Q^s$ and raises up to 4.52 $\pm$ 0.1 seconds at $\Delta = Q^s/64$. On the other hand, the computation time of the `companion` algorithm is considerably greater than the one reported using `cyclic`; for instance, when $\Delta = Q^s$ it is 552.02 $\pm$ 8.46 ms, which is comparable to the one obtained for `cyclic`, but for $\Delta = Q^s/64$, the time reaches 522.43 $\pm$ 1.49 seconds, namely two orders of magnitude greater than the result obtained with `cyclic`.

The probability computed by `analytic` is very close to the one of the numeric algorithms (either `cyclic` or `companion`) for $\Delta = Q^s$, while the computation time is several orders of magnitude below, remaining almost invariant around 542.21 $\pm$ 5.27 ms for any choice of Δ. In our experience with different distributions (both synthetic and experimental) the choice of $\Delta = Q^s$ has consistently produced an acceptable performance.

It is important to point out that the probability of respecting the deadline estimated by the `analytic` method is always conservative with respect to the `cyclic` or `companion` methods. This pessimism, along with the short time required to obtain such result, could be an advantage in certain situations. For instance, it could be used in an online admission

Table 4.1: Probability of respecting a deadline equal to the task period for different values of the bandwidth. The difference in the probabilities is evident for low values of the bandwidth. When the assigned bandwidth is increased, the gap is reduced.

Bandwidth	40%	48%	56%	64%	72%	80%
`analytic`	0.0000	0.6605	0.8771	0.9323	0.9536	0.9682
`cyclic`	0.3764	0.8178	0.9179	0.9579	0.9680	0.9813

test to guarantee the probability of respecting the deadline for a task. Another possibility could involve to use the `analytic` method as a first step in an optimisation procedure. In this strategy, the `analytic` method could be used to estimate a fast initial conservative approximation that will be further refined towards a more accurate value by applying the `cyclic` method.

Behaviour with Changing Bandwidth

In order to compare the accuracy of the `analytic` method against the numeric solutions (`cyclic`) for different bandwidths, we considered the same task from the previous experiments. The task period and server period were $T = 200$ ms and $T = 50$ ms respectively while the budget Q^s was changed so that the resulting bandwidth varies in the range [40%, 80%]. The granularity was fixed to $\Delta = 240\,\mu$s for `cyclic` to achieve a good approximation and to $\Delta = Q^s$ for the `analytic` solution.

The results reported in Table 4.1 show an important gap between `analytic` and `cyclic` for small values of the bandwidth. The gap is significantly reduced for bandwidth greater than 56%/64%. Smaller values of the bandwidth produce a probability level below 0.8, which is not acceptable for most real–time applications. The reason for the improvement of the analytic bound when the bandwidth increases is probably due to the fact that the system recovers more easily from large delays and this alleviates the impact of the conservative simplifications that underlie the analytic model.

The changes in the bandwidth do not affect the computation time required by PROSIT to obtain these results, as presented in Table 4.2, where is reported the average time and the 95% confidence interval for 50 executions of the stochastic analysis for the two methods: `analytic` and `cyclic` for different values of the bandwidth. Indeed, the main factor for the time required for the analysis is given by the size of the probability transition matrix P, presented in Equation (4.1). As discussed in the previous section, the scaling factor Δ has a direct impact in the matrix size while the assigned bandwidth has a smaller impact.

Table 4.2: Time required by PROSIT to calculate the probability of respecting a deadline equal to the task period for different values of the bandwidth. The required time remains stable and it is not affected by the variations of the assigned bandwidth.

Bandwidth	**Time**	
	`analytic` [ms]	`cyclic` [s]
40%	540.82 ± 4.76	187.48 ± 0.44
48%	541.13 ± 4.00	180.15 ± 0.15
56%	539.26 ± 3.84	175.12 ± 0.34
64%	542.11 ± 3.91	177.05 ± 0.34
72%	548.44 ± 6.90	178.38 ± 0.37
80%	540.12 ± 4.39	176.65 ± 0.38

4.3.2 The Lane Detection Application

As a second test case, we have considered a robotic vision program that identifies the boundaries of a lane and estimates the position of a mobile robot by using a web–cam mounted on the chassis of the robot [46]. The computation was carried out using a WandBoard[4] running Ubuntu. The version of the Kernel used (4.8.1) implements the CBS algorithm (under the name of SCHED_DEADLINE [70] policy) alongside the standard POSIX real–time fixed priority policies (SCHED_FIFO and SCHED_RR).

The robot executed 30 different paths across an area delimited by a black line. For each run, we have captured a video stream containing the line. The data sets roughly consisted of 2500 frames each and were later used for multiple off–line executions of the vision algorithm. A first group of 10 executions for each data set was performed with the algorithm executed in a task running alone and scheduled with the maximum real–time priority (99 for SCHED_FIFO). The maximum execution time of real–time tasks per period on Linux (RT throttling) was set to 95%. This allowed us to collect statistics of the computation time associated with the data set.

In a second group of executions, we have replicated a real–life condition. The vision algorithm was, in this case, executed in a periodic task processing a frame every $T = 100$ ms. The task was scheduled using SCHED_DEADLINE, with server period $T^s = 25$ ms and with different choices of the bandwidth in the range [15%, 30%]. For each data set and for each choice of the bandwidth, we repeated 10 executions recording the probability of respecting the deadline, expressed as the ratio between the number of jobs that finished before the deadline over the total number of jobs.

[4] `www.wandboard.org`

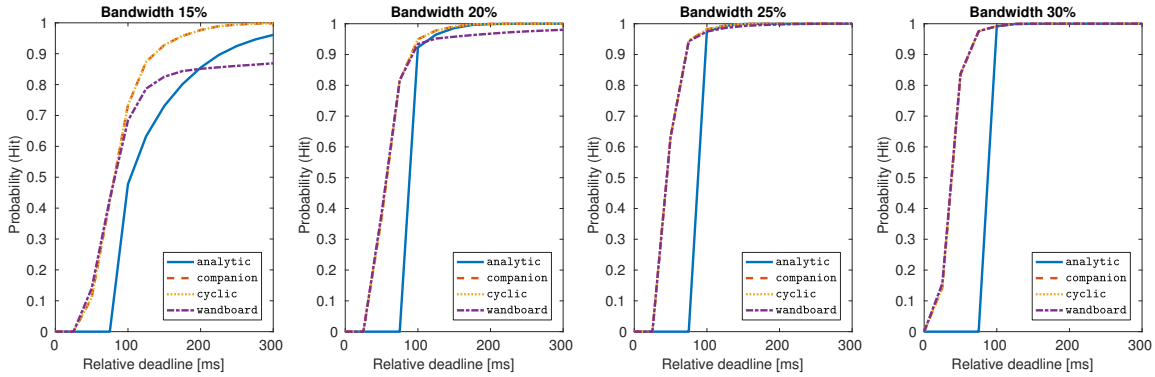

Figure 4.4: Comparison of the probability of respecting the deadline for different analysis methods and different values of assigned bandwidth. For low values of the bandwidth, there is a considerable gap, caused by the violation of the i.i.d. assumption, between the proposed methods and the experimental results. The problem is no longer observed when the assigned bandwidth is increased.

The probability averaged through the 10 executions was compared with the one obtained by PROSIT, executed with different solution methods and with the distribution estimated from the data set as input.

In accordance with the obtained in the previous experiments, the computation time required by PROSIT to produce these results was pretty much aligned among the different bandwidths. In particular, to solve the analytic bound (`analytic`), PROSIT required 58.63 $\pm$ 0.25 ms. The solution of the proposed numeric solution (`companion`) required 213.18 $\pm$ 1.22 seconds, while `cyclic` took 16.11 $\pm$ 0.07 seconds.

Figure 4.4 reports the probability of respecting the deadline for 4 representative choices of the bandwidth. The chosen scaling factor were $\Delta = 50\,\mu$s for `cyclic` and `companion` and $\Delta = Q^s$ for `analytic`. As expected, the four different values of bandwidth shown in the four sub–plots produced different probability of meeting the deadline.

As we observe in the plot, in the cases when the task is guaranteed enough bandwidth (more than 25%), the numeric algorithms (either `cyclic` or `companion`) produce a close match with respect to the experimentally obtained probabilities (`wandboard`) with an error smaller than 1%. On the other hand, the analytic bound performs well for deadlines greater than or equal to the task period, but its accuracy is not that good for deadline smaller than the period. It is worth to note that `analytic` provides a more conservative guarantee.

The situation is quite different when the assigned bandwidth is below the 25%. In this case, the proposed methods overestimate the probability of respecting the deadline, which can lead to several problems when the system executes under real conditions.

It is important to note that the gap in this overestimation increases significantly when reducing the assigned bandwidth. For instance, consider the left plot in Figure 4.4 representing a bandwidth of 15% ($Q^s = 3.75$ ms). In this case, the experimentally obtained probability (`wandboard`) is 0.68 for a deadline equal to the task period; while the proposed analysis (`companion`) reports a probability of 0.73 and the analytic bound (`analytic`) indicates a probability of 0.48.

As mentioned before, for this experiment the mathematical assumptions required to apply the proposed methods are not satisfied, hence the results can be considered only as a fair approximation. In particular, we observe that the vision algorithm iteratively builds upon previous results to produce the estimate. This characteristic introduces a strong correlation structure in the process that disrupts the assumptions required for an exact application of the method.

Even if, by adjusting the scheduling parameters of the task, the level of approximation could be acceptable when dealing with soft real–time systems, it remains the fact that for certain applications, the i.i.d. assumption is not verified. Therefore, the main contribution of this thesis, discussed in Chapter 5, is focused towards the development of a strategy that, considering the correlation in the computation times of such applications, is able to provide tight conditions for the probabilistic analysis of soft real–time systems.

Chapter 5

Non I.I.D. Computation Times

Several techniques for probabilistic guarantees exist for resource reservation schedulers and are based on the assumption that the process describing the application is independent and identically distributed (i.i.d.). However, as seen in the experimental validation presented in Section 4.3.2, there exists certain real–life applications for which this assumption does not hold.

In order to tackle these situations, it is necessary to devise an alternative model for the computation times that accounts for the characteristic correlation structure of these applications.

This chapter introduces a Markovian model for the computation times of these type of applications. Additionally, it proposes a technique based on the theory of hidden Markov models to extract the structure of the model from the observation of a number of execution traces. Moreover, it presents a methodology for the computation of the probability of deadline miss for a periodic real–time task scheduled by a resource reservation algorithm. This methodology is adapted from the standard techniques for probabilistic guarantees.

Our experimental results, obtained from a relevant set of state–of–the–art robotic applications, reveal a very good match between the proposed theoretical findings and the experiments. This good match is also reached when using synthetically generated distributions.

5.1 Randomized Methods

In the large class of robotic applications that can benefit from randomised methods, two deserve a special mention:

- Computer vision algorithms used to extract information of interest from a scene captured using on–board cameras,

- Path planning.

The popular RANSAC algorithm and its numerous derivatives [44] fall in the class of computer vision algorithms. This algorithm has been invented to interpret the sensed data in terms of predefined models, which typically correspond to known objects or landmarks.

Classic algorithms (e.g., based on least square analysis) consider all the data presented to the algorithm and are not able to single out and reject gross deviations from the model. Single gross deviations correspond to unexpected findings and are frequently encountered when travelling across an unknown environment. They are considered as "poisoned points" also for heuristic algorithms and can significantly impair the scene analysis.

To address the problem, the RANSAC algorithm proposes to select random points in the scene, instantiate the model on a subset of the data and measure the deviation. The procedure is repeated until a good consensus is reached between the selected subset and the model. This paradigm is easy to understand and implement, and it usually delivers excellent results in extracting the meaningful information from the scene.

In the class of the planning algorithms we find popular randomised methods such as Rapid–Exploring Random Tree (RRT) [64], or its recent development called RRT* [58]. The path planning problem is about finding a collision–free path that connects two points in the work space such that its curvature is required to be compatible with the kinematic and the dynamic constraints of the robot.

Both algorithms follow an iterative approach using a randomised search. For instance, RRT constructs two trees: one starting from the origin and one from the destination. In each iteration, a random number of points are selected and connected to the ones found at the previous step (cancelling the points that would determine a collision). The procedure stops when the two trees intersect. The algorithm selects the path along the tree with the minimum costs using a simple greedy heuristic. The advantage of using randomised algorithms is that we do not need any prior knowledge on the environment (e.g., concerning the presence of obstacles), which is "discovered" during the construction of the trees.

Vision and planning algorithms have to be executed in real–time while the robot is on the move. As an example, for a mobile robot of the size of a small vacuum cleaner, moving in a crowded space at 0.5 m/s, the planning can be safely executed three times per second (once every 400 ms).

The vision algorithm is used to localise and, ultimately, to control the vehicle. Therefore, it needs to produce the data more frequently. Moving at the moderate speed of our example, 200 ms could be an acceptable sampling time. Clearly, if the speed of the robot increases, so the sampling frequency will also need to increase. Since it is a known fact

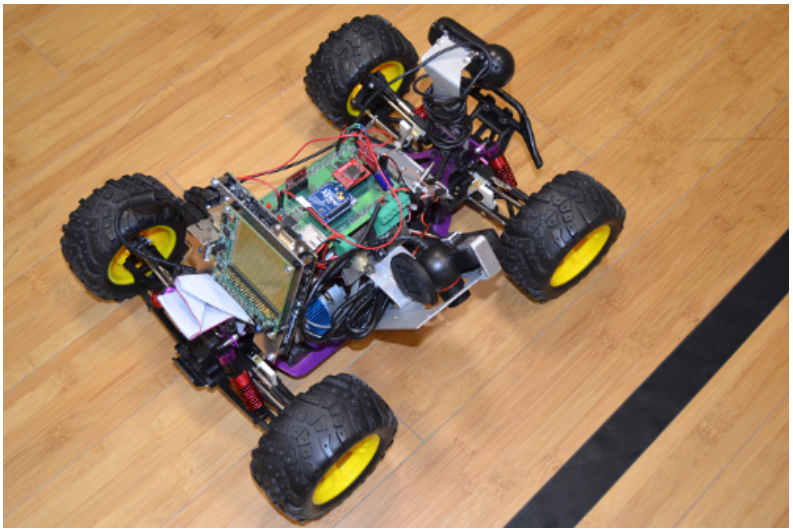

Figure 5.1: The robotic car used for one set of experiments. The lateral camera is used to capture the image and the lane detection algorithm is executed on the on–board platform.

that occasional timing failures can be acceptable for this type of systems [47, 27], we can easily trade such a possibility for a higher "average" sampling rate. However, an accurate assessment of the probability of these occasional failures is key to a correct design.

We now discuss in detail the applications that we will use as case study for this thesis and focus on the evolution of the computation time.

5.1.1 The Lane Detection Algorithm

The lane detection algorithm summarised below is used in some mobile robot applications developed in our department (see Figure 5.1). The goal of the algorithm is to determine the position of the robot with respect to a line delimiting the lane (see Figure 5.2). More specifically, the expected output of the algorithm is the distance y_p of the centre of the vehicle from the line and the angle θ_p between the longitudinal axis of the car and the line.

The pair (y_p, θ_p) can be used to control the lateral position of the vehicle in the lane. Unsurprisingly, if the vehicle travels at a high speed, the sampling period has to be rather small (although occasional deadline misses can be tolerated).

The algorithm is described in previous literature [46]. For the reader's convenience, we summarise here the main steps of the work flow:

1. The first step is a preprocessing of the image frame captured by the camera, in which the image size is reduced and the edges are detected to identify any relevant feature of the image (e.g. the line to be followed). Looking at the sample image in

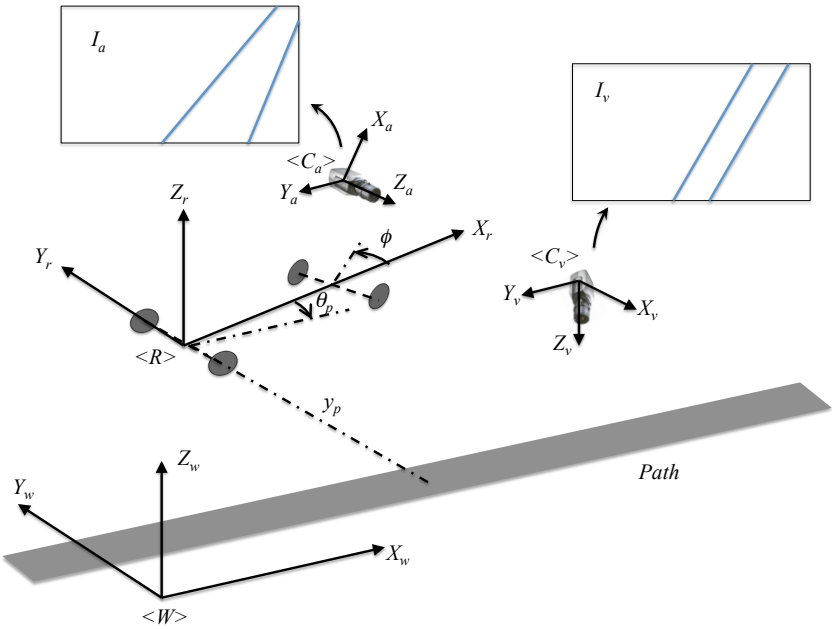

Figure 5.2: Setup of the vision system. The actual camera (looking at the road) captures a frame with a certain perspective (I_a) and the virtual camera (looking from above) "produces" a frame (I_v) where the perspective is corrected.

Figure 5.3 and moving from left to right, we can observe the image resizing and the canny filtering that extracts the edges, filtering out unnecessary elements.

2. Using an Inverse Perspective Mapping (IPM) technique, the scene observed by the "virtual" camera, flying above the scene, is reconstructed from the scene viewed by the actual camera. For the sample image in Figure 5.3, this corresponds to the fourth step. In the view from above, some elements of the frame are missing because they are simply not contained in the image captured from the camera due to the perspective effect.

3. By means of a RANSAC–based estimation algorithm, the position and orientation of the parallel lines in the "virtual" image are detected. The parallel lines sought in the image have to comply with an a priori model (e.g., the distance between the lines is known). The random search rules out all lines that receive a low compliance score and eventually comes up with the couple of parallel lines that are the closest to the model.

4. Finally, the position and orientation of the robot with respect to the actual line is computed. This information is then used to control the robot while moving in the

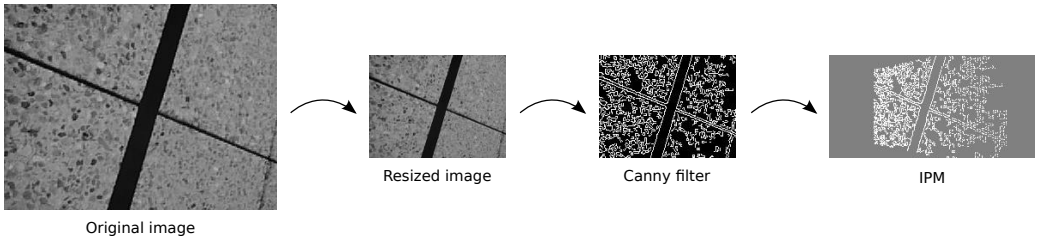

Figure 5.3: Sequential analysis of the image performed by the lane detection algorithm. The captured image is scaled down and filtered, then the Inverse Perspective Mapping technique reconstructs the "virtual" image from the "actual" image. After that, the RANSAC–based algorithm estimates the position and orientation of the robot with respect to the lane.

environment.

The algorithm described above is suitable for real–time implementation even on a cheap hardware platform. For instance, we have compiled the algorithm for an ARM 9 platform and executed it on a WandBoard[1] mounted on the mobile robot. The robot executed a linear path in the laboratory and its task was to follow a black ribbon unfolded on the floor and used as a lane delimiter.

In Figure 5.4, we report an excerpt of the trace of the computation times for a sequence of 900 jobs. As we can see, the computation time fluctuates but it has recognisable trends, which are revealed by the moving average of 25 samples drawn in red and superimposed on the trace. Such trends are obviously reflected into the autocorrelation function plotted on the right part of the figure. By inspecting the autocorrelation plot, we can see that a sample of the computation time is not only strongly correlated with the previous ones, but the correlation extends quite far into the past (the normalised value of the correlation remains greater than 20% even 20 samples behind).

In other words, the computation time of a job is correlated to the computation times of previous jobs, hence it cannot be described using a simple probability distribution function $C(c) = \mathbf{Pr}\{c_j = c\}$, as done in previous works [3, 8, 84]. As a consequence, more advanced mathematical techniques have to be used to properly and more precisely describe the computation times.

[1] `www.wandboard.org`

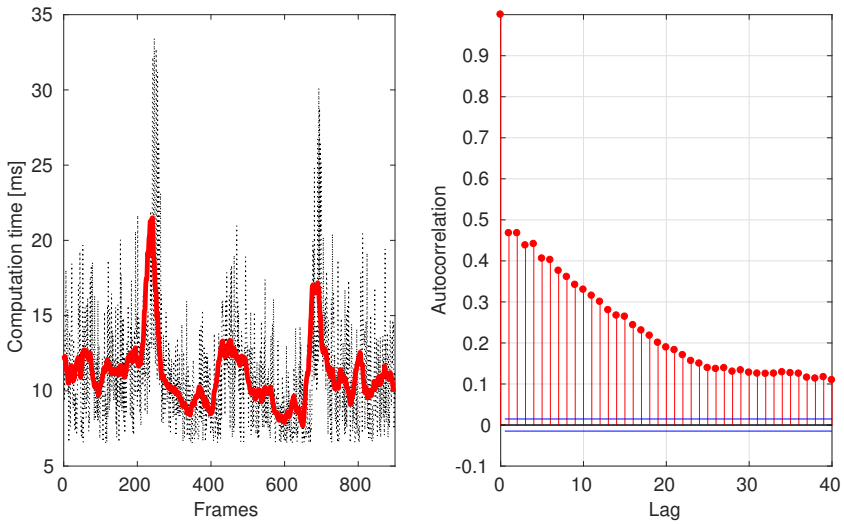

Figure 5.4: The left plot shows the computation times (dotted line) of the images taken by the robot while navigating the "clean" environment along with a superimposed (solid line) moving average to highlight the trend. The right plot shows the autocorrelation function of those computation times for different "lags" or time instants.

5.1.2 The Reactive Planning Algorithm

In the context of the European Project ACANTO[2], our department is involved in the development of a robotic walking assistant. The robot, called "FriWalk" (see Figure 5.5), can guide the user along a path and suggest a velocity by various means. Reasonably, the operation environment can be a crowded one, hence it is expected that the robot detects and avoids either fixed obstacles or pedestrian in the surroundings.

The FriWalk is a standard commercial walking aid endowed with sensing abilities to understand the surroundings, with communication abilities to connect to cloud services, with planning abilities to produce safe paths in the environment and with a guidance system to constrain the platform to a desired path as the user provides thrust, preserving safety and maximising comfort [30, 82].

The goal of the reactive planning algorithm is to modify the originally planned path in order to avoid pedestrians in the surroundings. The algorithm relies on a very accurate model to predict the motion of each pedestrian; for instance, the Headed Social Force Model (HSFM) [42]. The possible trajectories for both the robot and the pedestrians are efficient to generate, collision free (up to a certain probability) and comfortable to follow.

The algorithm is described in previous literature [17]. For the reader's convenience, we summarise here a possible scenario explaining the main steps of the algorithm:

[2]A CyberphysicAl social NeTwOrk using robot friends. http://www.ict-acanto.eu/acanto

Figure 5.5: The robotic walker, *FriWalk*, with the sensing system and embedded hardware.

1. The user of the robotic walker wants to go from point A to point B, hence a high level planner [30] produces an optimal *global path* (GP) that avoids static obstacles [16].

2. While moving, the FriWalk encounters an unforeseen obstacle along the planned trajectory. The HSFM predicts the possible future motion of human obstacles generating a number of possible trajectories. Each trajectory is associated with a possible destination, velocity profile and the probability that it will be taken by the pedestrian.

3. Then, the planning algorithm considers alternative paths for the robot, and for each computes the intersection with the possible curves taken by the pedestrian. By considering the probability associated with each of these curves, the planner computes the total probability of a collision and uses it to identify the trajectory with acceptable collision probability that minimises the deviation from the GP.

4. If a replanning is requested, the algorithm selects two points Q_0 and Q_2 on GP; the replanned trajectory will depart from Q_0 and will rejoin it at Q_2. The algorithm seeks a new point Q_1 in the proximity of the obstacle to pivot on to find the best trajectory, as presented in Figure 5.6.

5. If the algorithm is not able to find a solution that satisfies the geometric or the

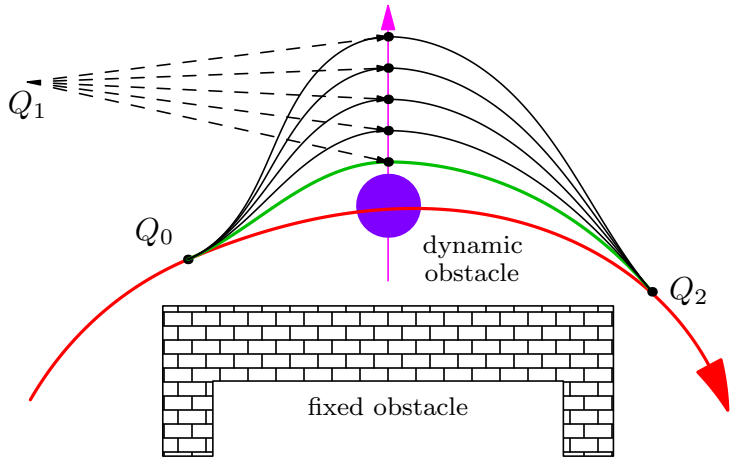

Figure 5.6: A sketch of the local re–planning method. In red the global trajectory that is no longer feasible because of the obstacle (purple circle). In green the optimal escaping manoeuvre, in black feasible candidates for different choices of Q_1 located deterministically aside from the obstacle trajectory.

dynamic constraints, the robot can apply is backup solution, namely slow down or stop on the spot and wait until the original plan is feasible to be followed.

The described algorithm is able to satisfy the initial requirements: 1) the re–planned trajectory is collision free to a reasonable extent; 2) the path is socially acceptable and comfortable to follow; and 3) the trajectory re–planning can be computed in real–time and does not require expensive hardware to be implemented.

Moreover, the computation times collected from the planner under different conditions corresponding to a different density of obstacles in the environment, show characteristics that make them suitable to be modelled as a MCTM: 1) Fluctuates, as is typical for randomised methods; 2) Depends on the complexity of the input data set; and 3) For a given operating condition, it is uncorrelated. Section 5.5 presents the results obtained by analysing the reactive planning application.

5.2 The Markov Computation Time Model

The randomised nature of the algorithms described in Section 5.1 generates a random computation time even for multiple executions on the same input data. However, as long as the input data set has the same complexity, the computation time will be of comparable magnitude. When the input data changes (e.g., due to a change in the scene), the computation time is likely to grow or decrease (although there will still be significant random fluctuations).

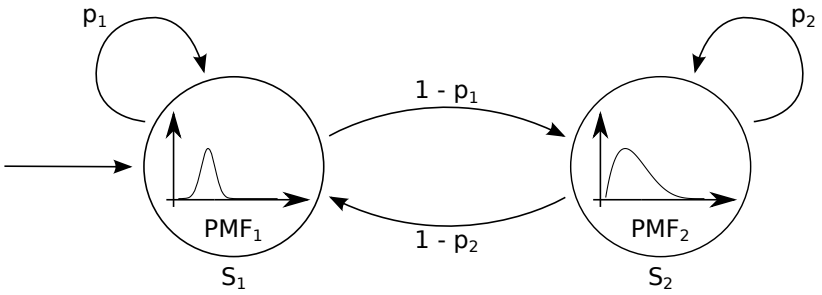

Figure 5.7: A mode change approach to model, as a Markovian process, the correlation in a stochastic process. The system starts in the state S_1 where the computation times are i.i.d. and described by PMF_1. With a given probability (p_1), the system will remain in this state. However, there is a non–zero probability $(1 - p_1)$ to change to state S_2, where the computation times are also i.i.d. and described by PMF_2. The resulting sequence of computation times, observed by the realisation of this Markov chain, are non i.i.d.

This switching behaviour introduces correlation in the stochastic process that models the computation time; hence, as previously noticed, it is not possible to describe the computation times of a task using a simple probability distribution.

A possible way to work around this issue is to find some kind of structure in the correlation between computation times, and to describe such a structure with some stochastic model.

The switching behaviour highlighted above suggests that it could be possible to describe the computation times within the limits of a tractable model by describing the different operating conditions of the system with a finite number of states (N). In each of these different modes, the computation time is described by a random variable; if the computation times for each mode are independent and identically distributed (i.i.d.), then it is possible to describe such random variables by associating a probability distribution to each state.

Figure 5.7 introduces a graphical representation of the proposed approach. The system starts in state S_1 where the computation times are described by an i.i.d process associated to PMF_1. With probability p_1, the system will remain in this state, however there is a chance for the system to "jump" to state S_2 where the computation times are associated to PMF_2. This switching behaviour continues and introduces the observed correlation in the computation times.

This idea is well expressed by the lane detection algorithm reported in Figure 5.8. On the left hand side, we observe an execution of the algorithm on a "clean" frame. On the

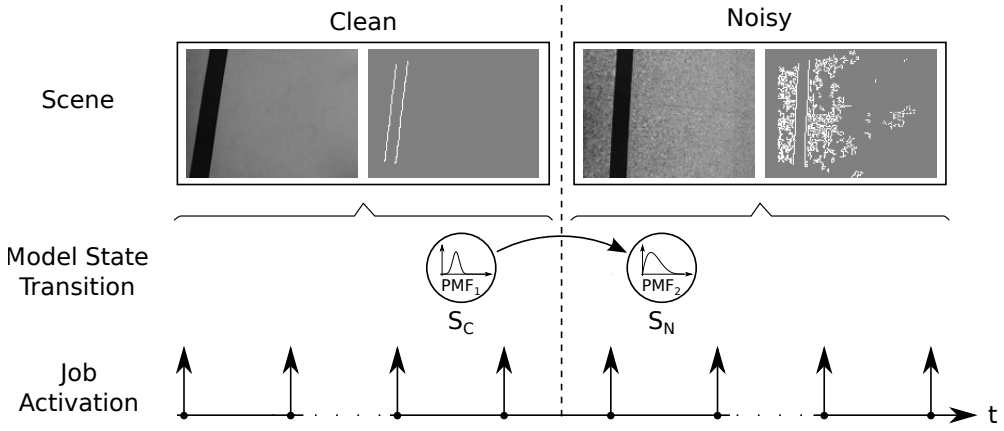

Figure 5.8: Example of scene variation with the corresponding mode change. The robot travels a certain environment characterised by two different types of floor conditions. For a number of activations, the robot stays in the "clean" scene, where the computation times are described by PMF_1. Suddenly, the robot changes scene (to the "noisy" one) with computation times described by PMF_2. This switching behaviour introduces correlation in the computation times.

right hand side, we observe the execution of the same algorithm on a "noisy" frame, in which the computation time required to extract the lines is presumably much higher.

If the transitions between the different system states can be defined as a Markov process, the model just described corresponds to a particular type of non i.i.d. process that occurs very frequently in the applications, under the name of *Markov Modulated Process* (MMP) [43]. To be precise, since the state of the Markov chain is not directly accessible or observable (only the computation times are observable), the model can also be defined as a hidden Markov model (HMM). HMMs are very popular in several disciplines, such as economics and biology [40].

In this thesis, we will show that a HMM can be used to model the evolution of the computation times for a large class of robotic applications, and that some probabilistic analysis developed for i.i.d computation times can be adapted to work with this non i.i.d model.

Given our specific application, we will name this model: *Markov Computation Time Model (MCTM)*. A precise definition is offered next.

Definition 2 *A Markov Computation Time Model is defined as the triplet* $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$*, where*

- $\mathcal{M} = \{m_1, \ldots, m_N\}$ *is the set of modes;*

- $\mathcal{P} = (p_{a,b})$, *with* a *and* $b \in \mathcal{M}$, *is the mode transition matrix. The element* $p_{a,b}$ *defines the probability that, at the next step, the system will be in mode "b"* ($m_j = b$) *given that, at the previous step, the system was in mode "a"* ($m_{j-1} = a$)*:*

$$p_{a,b} = \mathbf{Pr}\left\{m_j = b \,|m_{j-1} = a\right\};$$

- $\mathcal{C} = \left\{C_{m_j} : m_j \in \mathcal{M}\right\}$ *is a set of distributions characterising the computation time in each mode. Each distribution is described by the Probability Mass Function (PMF) since the computation times can only take values that are multiples of the processor clock. Therefore, with this notation,* $C_{m_j}(c)$ *represents the probability that the computation time is equal to* c *when the system executes in mode* m_j*.*

Hence, every mode has an associated probability distribution for the computation times, and the mode transitions determine changes in the distribution of the computation time. We will make the following (reasonable) hypotheses:

Assumption 1 *For every job* J_j*, the computation time* c_j *is a random variable described by the distribution* C_{m_j}*, which only depends on the current mode* m_j *of the MCTM. The transitions between modes happen according to the probabilities* $p_{a,b}$*. Furthermore, the "mode change" event is independent both from the current computation backlog and from the computation time required by the previous execution.*

Remark 3 *The process governing the mode change is typically rooted in the physics of the system (e.g,, a robot travelling in a space and encountering an obstacle). Therefore, the assumed independence between the length of the computation time and the mode change is reasonable.*

What is more, the mode change is generally asynchronous with respect to the start of the job, and it could, in reality, take place in the inter–sample period between the arrival of two jobs. However, for the purposes of the system, the mode change event has an effect on the computation time only in the next period (when a new sample is collected). Therefore, the synchronous assumption made above is also perfectly reasonable.

5.3 Stochastic Analysis

In this section, the analysis of the timing evolution of a periodic task, with period T and computation time described by a MCTM, when it is scheduled through a CPU reservation with parameters (Q^s, T^s), is shown. Assuming that the parameters describing the MCTM ($\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$) are known, it is shown how the dynamic evolution of a CBS scheduler can

be described by a Markov chain. Then, how to efficiently compute the steady state distribution of the probability of the states of the Markov chain, is discussed. Finally, how to compute the distribution of the response time of the task using the steady state distribution of the states of the Markov chain, is presented.

5.3.1 Dynamic Model

As presented in Section 3.3, the timing evolution of a CPU reservation can be represented by a Markov chain with an infinite number of states. Moreover, Section 3.3.1 introduces δ_j as an upper bound for the response time of the job and Equation (3.4) defines the dynamic evolution of δ_j.

Introduce the symbols $c_{m_j}^{\min}$ and $c_{m_j}^{\max}$ to denote the minimum and the maximum computation time for mode m_j: $c_{m_j}^{\min} = \min\{c \mid C_{m_j}(c) \neq 0\}$ and $c_{m_j}^{\max} = \max\{c \mid C_{m_j}(c) \neq 0\}$, with $m_j \in \mathcal{M}$. Let $c^{\min}$ and $c^{\max}$ be the minimum and the maximum computation time for all the different modes: $c^{\min} = \min\left\{c_{m_j}^{\min}\right\}$ and $c^{\max} = \max\left\{c_{m_j}^{\max}\right\}$. Since the computation time varies in the bounded set $\{c^{\min}, \ldots, c^{\max}\}$ regardless of the mode of the MCTM, the value of v_j will be lower bounded by $c^{\min}$.

For the case of computation times described by an i.i.d stochastic process, the model in Equation (3.4) represents a DTMC, where the states are determined by the possible values of v_j, and the transition probabilities by the PMF of the computation time. On the other hand, for the case of MCTM computation times, the model still represents a DTMC but it is necessary to extend the definition of the state in order to consider the operating mode.

Therefore, for the j^{th} job, the state of the system is captured by the pair (m_j, v_j). Let us define the events $\mathcal{M}_g(j) = \{m_j = g\}$, meaning that the system is in mode "g" at job j, and $\mathcal{V}_h(j) = \{v_j = c^{\min} + h\}$, meaning that the workload is equal to $c^{\min} + h$ at job j, with $g \in \mathcal{M}$ and $h \in [0, \ldots, \infty[$ representing the total backlog accumulated until the previous job (v_{j-1}). Figure 5.9 shows a graphical representation of a DTMC describing a resource reservation where the computation time of the task is described by a Markov Computation Time Model.

Under this consideration, the state of a MCTM can be formally defined as follows:

Definition 3 *The state of a MCTM, $\mathcal{S}_{g,h}(j)$, is defined as the intersection of the previously defined events: $\mathcal{S}_{g,h}(j) = \mathcal{M}_g(j) \wedge \mathcal{V}_h(j)$. In plain words, the event $\mathcal{S}_{g,h}(j)$ can be expressed as "the state (m_j, v_j) has the value $(g, c^{min} + h)$".*

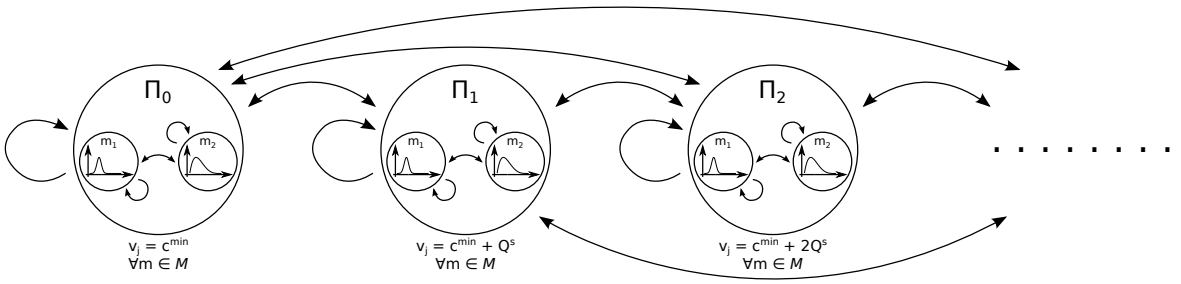

Figure 5.9: Graphical representation of the dynamic evolution of the states of the system, modelled as an infinite queue. The states are given by the possible values (infinite) of the workload v_j and the operating mode m_j. The probability of such values is given by $\pi_{g,\,h}(j) = \mathbf{Pr}\left\{m_j = g \wedge v_j = c^{\min} + h\right\}$ and the evolution of the vector $\Pi = [\Pi_0, \Pi_1, \ldots]$, with $\Pi_h(j) = [\pi_{1,\,h}(j)\, \pi_{2,\,h}(j) \ \ldots \ \pi_{N,\,h}(j)]$ can be described using the standard notation of the Markov chains.

Based on Definition 3 and Equation (3.2), it is possible to write:

$$\begin{aligned}\mathbf{Pr}\left\{\mathcal{S}_{g',\,h'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\} &= \mathbf{Pr}\left\{\mathcal{M}_{g'}(j) \wedge \mathcal{V}_{h'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\} \\ &= \mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\} \cdot \\ &\quad\ \mathbf{Pr}\left\{\mathcal{V}_{h'}(j)\,|\mathcal{S}_{g,\,h}(j-1) \wedge \mathcal{M}_{g'}(j)\right\}.\end{aligned} \tag{5.1}$$

As a direct application of Definition 3, the first term of Equation (5.1) can be written as:

$$\mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\} = \mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{M}_{g}(j-1) \wedge \mathcal{V}_{h}(j-1)\right\}. \tag{5.2}$$

In view of Assumption 1, the mode change is independent of the computation time of the previous job. Hence, it is possible to write Equation (5.2) as:

$$\begin{aligned}\mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\} &= \mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{M}_{g}(j-1)\right\} \\ &= \mathbf{Pr}\left\{m_j = g'\,|m_{j-1} = g\right\} \\ &= p_{g,\,g'}.\end{aligned} \tag{5.3}$$

The second term of Equation (5.1) can be written as:

$$\begin{aligned}&\mathbf{Pr}\left\{\mathcal{V}_{h'}(j)\,|\mathcal{S}_{g,\,h}(j-1) \wedge \mathcal{M}_{g'}(j)\right\} \\ &= \mathbf{Pr}\left\{\mathcal{V}_{h'}(j)\,|\mathcal{M}_{g}(j-1) \wedge \mathcal{V}_{h}(j-1) \wedge \mathcal{M}_{g'}(j)\right\} \\ &= \mathbf{Pr}\left\{v_j = c^{\min} + h'\,\middle|\, m_j = g' \wedge m_{j-1} = g \wedge v_{j-1} = c^{\min} + h\right\}.\end{aligned} \tag{5.4}$$

In view of Assumption 1, the process modelling the sequence c_j of the computation times depends neither on v_j nor on m_{j-1}, but solely on m_j. Thereby, taking into account Equation (3.4), it is possible to write Equation (5.4) as:

$$\begin{aligned}
&\mathbf{Pr}\left\{v_j = c^{\min} + h' \,\middle|\, m_j = g' \wedge m_{j-1} = g \wedge v_{j-1} = c^{\min} + h\right\} \\
&= \mathbf{Pr}\left\{\left[c^{\min} + h - n \cdot Q^s\right]^+ + c_j = c^{\min} + h' \,|m_j = g'\right\} \\
&= \mathbf{Pr}\left\{c_j = c^{\min} + h' - \left[c^{\min} + h - n \cdot Q^s\right]^+ |m_j = g'\right\} \\
&= \begin{cases} C_{g'}(c^{\min} + h') & \text{if } h \leq n \cdot Q^s - c^{\min} \\ C_{g'}(h' - h + n \cdot Q^s) & \text{otherwise} \end{cases}.
\end{aligned} \tag{5.5}$$

Note that, since we are assuming a periodic task, $n = T/T^s$. As a result, by combining Equation (5.3) and Equation (5.5), we have:

$$\mathbf{Pr}\left\{\mathcal{S}_{g',h'}(j) \,|\mathcal{S}_{g,h}(j-1)\right\} = \begin{cases} p_{g,g'} C_{g'}(c^{\min} + h') & \text{if } h \leq n \cdot Q^s - c^{\min} \\ p_{g,g'} C_{g'}(h' - h + n \cdot Q^s) & \text{otherwise} \end{cases}. \tag{5.6}$$

We can now introduce a vector containing all the probabilities of the different states. Let $\pi_{g,h}(j)$ be defined as $\mathbf{Pr}\left\{\mathcal{S}_{g,h}(j)\right\} = \mathbf{Pr}\left\{\mathcal{M}_g(j) \wedge \mathcal{V}_h(j)\right\}$. In plain words, $\pi_{g,h}$ represents the probability that at the j^{th} job the mode m_j be "g" and the workload v_j be $c^{\min} + h$. Introduce the vector $\Pi_h(j) = [\pi_{1,h}(j) \;\; \pi_{2,h}(j) \;\; \ldots \;\; \pi_{N,h}(j)]$, which essentially represents the probability for the workload v_j to be $c^{\min} + h$ for all the possible modes of the MCTM.

We can stack the vectors $\Pi_h(j)$ into a single probability vector $\Pi(j)$ composed by an infinite number of elements:

$$\Pi(j) = [\Pi_0(j) \;\; \Pi_1(j) \;\; \Pi_2(j) \;\; \ldots]. \tag{5.7}$$

The evolution of the vector $\Pi(j)$, presented in Equation (5.7), can be described using the standard notation of the Markov chains [26]: $\Pi(j) = \Pi(j-1) \cdot P$.

The matrix P is called transition matrix and describes the evolution of the probability of finding the system in each state starting from an initial distribution. In our case, the vector $\Pi(j)$, presented in Equation (5.7), is made of an infinite number of elements, and so will be the matrix P.

By applying Equation (5.6), we can see that the matrix has a recursive structure. To this end, introduce the notation $\alpha_{g,h} = C_g(c^{\min}+h)$, $R = c^{\max} - c^{\min}$, and $S = (n \cdot Q^s) - c^{\min}$.

The matrix has the following block structure:

$$P = \begin{bmatrix} P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots \\ P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ \mathbf{0} & P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \dots & \\ \mathbf{0} & \mathbf{0} & P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \dots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{5.8}$$

where

$$P_e = \begin{bmatrix} p_{1,1} \cdot \alpha_{1,e} & p_{1,2} \cdot \alpha_{2,e} & \dots & p_{1,N} \cdot \alpha_{N,e} \\ p_{2,1} \cdot \alpha_{1,e} & p_{2,2} \cdot \alpha_{2,e} & \dots & p_{2,N} \cdot \alpha_{N,e} \\ \dots & \dots & \dots & \dots \\ p_{N,1} \cdot \alpha_{1,e} & p_{N,2} \cdot \alpha_{2,e} & \dots & p_{N,N} \cdot \alpha_{N,e} \end{bmatrix},$$

and $\mathbf{0}$ denotes a block of zero of size $N \times N$.

5.3.2 Computation of the Steady State Probability

A few observations are in order:

1. Each row of the block matrix has at most $R+1$ non–zero blocks. Indeed, each new block is associated with a different value of the computation time and such values run from $c^{\min}$ to $c^{\min} + R = c^{\max}$;

2. As a consequence of Equation (5.6), starting from a value of the workload h smaller than or equal to S, the probability does not change with h but only depends on the workload h' after the transition; therefore, the first $S+1$ blocks of rows are repeated;

3. As a consequence of the same equation, starting from a value of the workload h greater than S, the probability is a function of the difference $h' - h$; therefore, from the block of rows $S+2$ onward, each block of row is obtained shifting the previous one to the right of one block and padding with a block of zeros $\mathbf{0}$.

In view of these considerations, the block matrix shown in Equation (5.8) has the periodic structure, shown in Equation (3.3), which is characteristic of a Quasi–Birth–Death Process (QBDP).

The four matrices C, A_0, A_1, A_2 describing the QBDP can be expressed in terms of the blocks P_e in Equation (5.8). Recalling the symbol $P(p_z : p_Z \,;\, q_z : q_Z)$, which denotes the sub–matrix obtained from P by extracting the block of rows from p_z to p_Z and the block

of columns from q_z to q_Z. For instance, $P(S+1 : S+2\,;\, 1 : 2)$ will denote the sub–matrix $\left[\begin{smallmatrix} P_0 & P_1 \\ \mathbf{0} & P_0 \end{smallmatrix}\right]$. By defining $H = \max\{S,\, R\}$, it is possible to set the matrices as described in Equation (3.8), reducing the transition matrix to the QBDP structure mentioned above.

After casting our system into the QBDP framework, we can capitalise on the rich body of results in the field. In particular, following the same line of arguments as in [84], it is possible to show that:

1. The system admits a steady state
$$\begin{aligned} \widetilde{\Pi} &= [\widetilde{\Pi}_0,\, \widetilde{\Pi}_1,\, \ldots] \\ &= \lim_{j\to\infty} \Pi(j) \\ &= \lim_{j\to\infty} [\Pi_0(j),\, \Pi_1(j), \ldots], \end{aligned} \tag{5.9}$$
where $\lim_{j\to\infty} \Pi_h(j) = \lim_{j\to\infty} [\pi_{1,\,h}(j) \;\; \pi_{2,\,h}(j) \;\; \ldots \;\; \pi_{N,\,h}(j)]$.

2. The steady state distribution $\widetilde{\Pi}$ is unique and independent from the initial distribution $\Pi(0)$,

3. The computation of the steady state probability can be done using the very efficient numeric solutions available in the literature. In particular, this thesis uses the Cyclic Reduction algorithm presented in [18] for the numeric solution, although the Logarithmic Reduction algorithm [66] is also suitable for this purpose.

5.3.3 Computation of the Distribution of the Response Time

Once again, from the steady state probability $\widetilde{\Pi}$, it is possible to recover the steady state distribution of the variable δ_j, which, as mentioned before, is an upper bound of the response time of the task. The steady state CDF can be reconstructed using Equation (3.10) and reported here:

$$\lim_{j\to\infty} \mathbf{Pr}\{\delta_j = \delta \cdot T^s\} = \sum_{h=((\delta-1)\cdot Q^s)+1}^{\delta\cdot Q^s} \lim_{j\to\infty} \mathbf{Pr}\{v_j = h\},$$

where, for the case of MCTM:

$$\lim_{j\to\infty} \mathbf{Pr}\{v_j = h\} = \begin{cases} 0 & \text{if } h < c^{\min} \\ \lim_{j\to\infty} \sum_{g=1}^{N} \mathbf{Pr}\{\mathcal{V}_{h-c^{\min}}(j) \wedge \mathcal{M}_g(j)\} & \text{otherwise} \end{cases}.$$

We can finally observe that

$$\lim_{j\to\infty} \sum_{g=1}^{N} \mathbf{Pr}\{\mathcal{V}_{h-c^{\min}}(j) \wedge \mathcal{M}_g(j)\} = \sum_{g=1}^{N} \lim_{j\to\infty} \pi_{g,\,h}(j).$$

As discussed in Equation (5.9), each element $\sum_{g=1}^{N} \lim_{j\to\infty} \pi_{g,h}(j)$ of the sum can be extracted from the steady state distribution $\widetilde{\Pi}$.

By using the steady state distribution of the process δ_j, presented in Equation (3.10), it is possible to guarantee probabilistic deadlines. Indeed, a probabilistic deadline (D, β) is guaranteed if $\lim_{j\to\infty} \mathbf{Pr}\{\delta_j \leq D\} = \sum_{h=1}^{D/T^s} \lim_{j\to\infty} \mathbf{Pr}\{\delta_j = h \cdot T^s\} \geq \beta$, where we made the natural assumption that D is chosen as a multiple of T^s.

5.4 HMM Identification

The analysis presented in Section 5.3 is based on the assumption that the parameters describing the MCTM, namely $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$, are known by the system designer; however, in a real scenario this is not the case. Generally speaking, the system designer should be able to collect, from the target platform, a long enough raw trace of computation times; hence it is necessary to devise procedures and methods for the identification of the underlying models.

In order to address this issue, the computation time traces will be analysed using methods borrowed from the theory of the hidden Markov models (HMM) [89] to identify if a MCTM exists that fits the data.

In particular, a tractable model can be derived if the process is assumed to behave as a Markov Modulated Process (MMP). This choice comes from the nature of the stochastic process at hand: the computation times of data–driven applications depend on the actual input data quality, which is very often consistent along consecutive executions. With this assumption, a MCTM is effectively modelled by means of a *hidden Markov model*.

In this case, the computation times are assumed to depend on an internal state m_j of the task that can assume a finite number of values, i.e. $m_j \in \mathcal{M} = \{m_1, \ldots, m_N\}$. If the task is in state m_j, then the execution time of the current job is distributed according to a probability mass function $\mathcal{C}_{m_j}(c) = \mathbf{Pr}\{c_j = c\}$, hence there is a different computation times probability distribution for each state. Of course, the internal state m_j cannot be directly measured and it is assumed to be modelled as a Markov chain, i.e. the probability to be in state m_k at time j only depends on the value of the state at time $j-1$ [25].

In our N–states HMM, the transition probabilities $p_{a,b} = \mathbf{Pr}\{m_j = b \,|m_{j-1} = a\}$ describe the stochastic transitions between the states. As is customary in the Markov chain literature [59], by denoting $\pi_j \in \mathbb{R}^n$, the row vector of probabilities of being in each state, we have immediately that $\pi_{j+1} = \pi_j \cdot P$, where $P = (p_{a,b})$ is the transition probability matrix. The probability distributions set $\mathcal{C} = \left\{C_{m_j} : m_j \in \mathcal{M}\right\}$ models all the probability mass functions associated to each state.

5.4.1 HMM Estimation

In order to analyse the (probabilistic) schedulability of the system, it is important to accurately describe the process of the computation times of a task using the MCTM from a given sequence of computation times. In the literature, this is known as the "*HMM Learning*" problem. Assuming the knowledge of the number of internal sates N, this problem can be addressed by using some well–known techniques such as the Baum–Welch algorithm [12]. The Baum–Welch algorithm is a form of Expectation Maximisation algorithm maximising the likelihood function L, which is defined as the probability for the identified HMM $\theta \triangleq (\mathcal{P}, \mathcal{C})$ to generate a sequence of measured values $c_0, \ldots, c_K$

$$L = \mathbf{Pr}\{c_0, \ldots, c_K \,|\theta\}.$$

The algorithm estimates $\mathcal{P}$ and $\mathcal{C}$ starting from two initial guesses that are iteratively refined. In each step of the iteration, the new estimations for $\mathcal{P}$ and $\mathcal{C}$ are computed based on the probability $\xi_j(a, b)$ for the HMM to be in state m_a at job J_j and in state m_b at job J_{j+1}, given the sequence of observed computation times $c_0, \ldots, c_{j+1}$

$$\xi_j(a, b) = \mathbf{Pr}\{m_j = a \wedge m_{j+1} = b \,|c_0, \ldots, c_{j+1}\}.$$

Based on this, a new estimation of $\mathcal{P}$ can be computed as

$$p_{a,b} = \frac{\sum_j \xi_j(a, b)}{\sum_j \sum_i \xi_j(a, i)}.$$

Although this algorithm works perfectly from the theoretical point of view, it can be subjected to numerical stability problems (underflow) when the number of observed computation times is too high, as noticed in [37]. This happens because $\xi_j(a, b)$ is computed based on the so called *forward probabilities* $\alpha_j(a)$, i.e. probability for the HMM to be in state m_a at job J_j

$$\alpha_j(a) = \mathbf{Pr}\{m_j = a \wedge c_0, \ldots, c_j\},$$

and *backward probabilities* $\beta_j(a)$, i.e. probability to see computation times $c_{j+1}, \ldots, c_K$ given that the task is in state m_a at job J_j

$$\beta_j(a) = \mathbf{Pr}\{c_{j+1}, \ldots, c_K \,|m_j = a\}.$$

The forward probabilities $\alpha_j(a)$ can be computed using the inductive expression

$$\alpha_j(a) = \sum_k \alpha_{j-1}(k) \cdot p_{k,a} \cdot C_{m_a}(c_j),$$

and the backward probabilities can be computed as

$$\beta_j(a) = \sum_k \beta_{j+1}(k) \cdot p_{a,k} \cdot C_{m_k}(c_{j+1}).$$

Since we have a lot of measured computation times, j can become large; hence, $\alpha_j(\cdot)/\beta_j(\cdot)$ are multiplied a large number of times for probabilities that are less than 1, thus becoming 0 due to underflow errors. This problem can be solved by modifying the Baum–Welch algorithm to use conditional probabilities instead of joint probabilities, as shown in [37].

Given a sequence of observed computation times, the revised Baum–Welch algorithm is able to estimate the transition matrix $\mathcal{P}$ and the probability distributions $\mathcal{C}$ if the number N of distinct internal states m_j is available. In other words, the dimension of $\mathcal{P}$ and the number of different PMFs $\mathcal{C}$ should be known upfront. Hence, the problem is how to estimate such a value. In order to correctly dimension N, we made use of a *cross–validation* approach for the likelihood [94].

Therefore, to estimate the number of states N we adopted a gradient–like approach:

1. Set $N = 1$, then execute the EM procedure and evaluate the cross–validated likelihood L;
2. Set $L^\star = L$;
3. Set $N = N + 1$, then execute the EM procedure and evaluate the cross–validated likelihood L;
4. If $L \leq L^\star$, go to step 2);
5. The optimal number of states is $N - 1$, with cross–validated likelihood $L^\star$.

It has to be noted that this algorithm is based on the experimental evidence, that increasing the number of states more than needed also increases the cross–validated likelihood. Although only a conjecture at this point, grounded to the theoretical results on gradient–based methods [25], this approach proved to be efficient in all the adopted test cases, as presented in Section 5.4.2 and Section 5.5.

Another remarkable feature is that the cross–validated likelihood can also be used to establish the maximum number of computation time measures to be used in the identification, again using a gradient–like algorithm. Even though more data means higher estimation accuracy, we found out that the number of measures to be analysed can be limited according to the observed improvement of the cross–validated likelihood gradient.

5.4.2 HMM Validation

Once the state transition matrix $\mathcal{P}$ and the output probability distributions $\mathcal{C}$ have been estimated, they can be used as an input for the analysis technique described in Section 5.3. However, it is first important to check if the identified MCTM correctly describes the

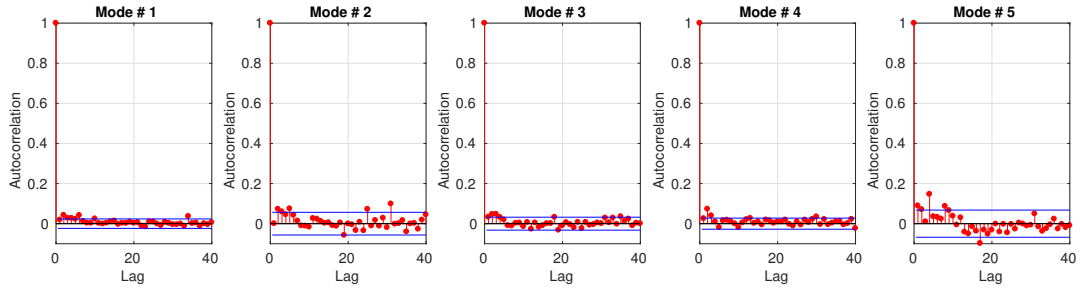

Figure 5.10: Autocorrelation function obtained from the 5 different modes in which the computation times presented in Figure 5.4 were classified. The strong correlation presented in the original sequence of computation times has disappeared. The 5 resulting sequences of computation times show a very low correlation.

computation times of the application. To do this, each observed computation time can be associated to a hidden state, so that the sequence $c_0, \ldots, c_K$ is split in N sub–sequences (one sequence per MCTM state), the independence of each sub–sequence can be tested, and the autocorrelations can be computed. This can be done by generating the most likely sequence of internal states for the identified MCTM given the computation times sequence $c_0, \ldots, c_K$. Such a problem is known as *HMM Decoding* problem in HMM literature, and can be easily solved by using the well–known Viterbi algorithm [48].

The proposed technique for the parameters identification and validation of the underlying MCTM has been implemented in the PROSIT[3] [99] tool, described more in detail in Chapter 6. As an example, the Baum–Welch algorithm has been fed with the sequence of computation times measured for the previously described robotic application (see Figure 5.4). This allowed us to identify a 5–states MCTM (trying to identify a MCTM with more than 5 states resulted in duplicated states).

The Viterbi algorithm has then been used to identify the most likely sequence of internal (hidden) states corresponding to the sequence of observed computation times, so that it has been possible to associate each computation time to one of the 5 hidden states of the identified MCTM. This allowed us to partition the sequence of computation times in 5 sub–sequences, to perform a numerical test for independence [73] on such sub–sequences, and to compute the autocorrelation for each one of them.

The results are presented in Table 5.1 and in Figure 5.10, and show that the computation times in each state are independent; hence, they can be correctly described by the identified output probability distributions, and the MCTM properly models the sequence of measured computation times.

The identification and validation processes can be time consuming. There are several

[3]https://bitbucket.org/luigipalopoli/prositool.git

Table 5.1: Results of the numerical independence tests for the 5 sub–sequences estimated by the Baum–Welch algorithm. A p–value greater than 0.01 allows us to accept the independence assumption.

State	z–statistic	p–value
1	−0.5450	0.2929
2	−1.3929	0.0818
3	−1.5830	0.0567
4	−1.2692	0.1022
5	−1.1088	0.1338

factors that can affect directly the time required by PROSIT to properly estimate the parameters. In particular, we have: the number of states of the underlying MCTM, the desired number of iterations for the Baum–Welch algorithm, and the number of computation time samples used for the process. By tuning those values, it is possible to decrease the whole time of the process.

In this case of the robotic vision algorithm, the number of states ranged from 2 to 6, the number of iterations was set to 100, and the number of samples was set to 18400. The time required by PROSIT to estimate the HMM parameters was 665.19 ± 4.57 seconds for the identification and 209.47 ± 0.04 ms for the validation.

5.5 Experimental Validation

In order to show the practical applicability of the presented approach, we have evaluated it on the real robotic applications presented in Section 5.1.

5.5.1 The Lane Detection Application

The first application that we are considering is the: *lane detection*, described in Section 5.1.1, where a robotic vision program is used to identify the boundaries of the lane, estimating the position and orientation of a mobile robot while moving in a predefined track.

Experimental Setup

The experimental setup consisted of a black ribbon placed on the floor creating a track of 37 meters long. The linear velocity of the robot was constant and set to 0.6 m/s. The

robot executed 20 laps in the track, allowing us to capture, with a frame rate of 30 fps, a video stream containing the ribbon. This data set roughly consisted of 18400 frames.

Moreover, in order to test the vision algorithm under different conditions, 2 distinct tracks with different floor characteristics, as presented in Figure 5.8, were created.

The first track was considered a "clean" one given that the lane detection algorithm was able to remove all the undesired elements from the image. On the other hand, the second track was considered a "noisy" track because most of the undesired elements remain in the image after the preprocessing step. As expected, the computation times of the different tracks differ considerably.

As a result, the experiments were performed on 2 sets (1 "clean" and 1 "noisy") of 18400 frames each. These sets were used as input for several off–line runs of the task executing the vision algorithm.

All these runs of the lane detection algorithm were carried out using a WandBoard[4] running Ubuntu. The version of the Kernel used (4.8.1) implements the CBS algorithm (under the name of SCHED_DEADLINE [70] policy) alongside the standard POSIX real–time fixed priority policies (SCHED_FIFO and SCHED_RR).

On the other hand, all the results from the stochastic analysis presented in this section have been measured on a Dell Precision T1700 equipped with an Intel Core i7 8–core processor operated at 3.6 Ghz and with 32 GB of RAM.

Experimental Results

In order to collect statistics of the computation time associated with each data set, the task running the vision algorithm, scheduled with the maximum real–time priority (99 for SCHED_FIFO) and the default RT throttling (95% of CPU load), was run 100 times for each data set. In this way, each run will consist of 18400 jobs, with its corresponding computation time, each one representing the processing of one frame from the data set.

This group of 100 runs was divided into:

- Training set, which was analysed using the HMM identification techniques presented in Section 5.4.1 to identify if a MCTM exists that fits the data, and
- Testing set, which was used to validate the results of the HMM training phase.

As mentioned in Section 5.4.1, the Baum–Welch algorithm was fed with different sequences of measured computation times taken from the training set. This process has been repeated multiple times for different numbers of modes (ranging from 2 to 6). The algorithm identified 5 different modes for the "clean" track, which present independence

[4] `www.wandboard.org`

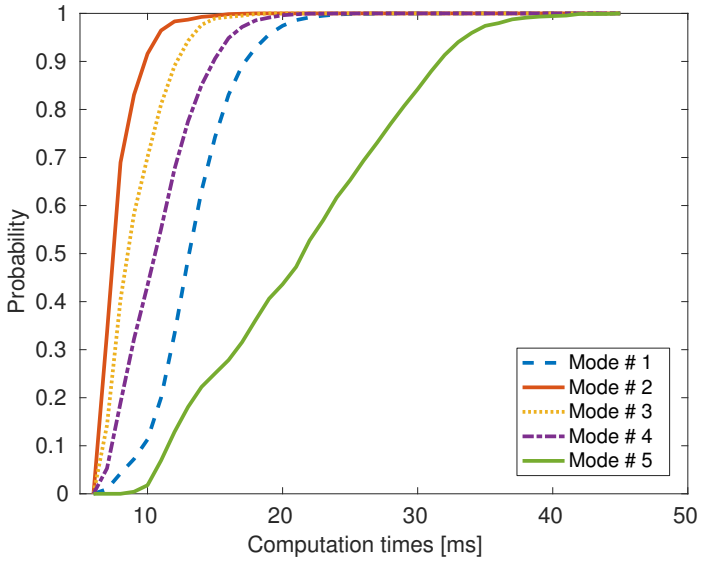

Figure 5.11: Empirical cumulative distribution functions for the 5 different operating modes. These modes were identified by applying the Baum–Welch algorithm to a sequence of computation times measured from the "clean" track.

in the computation times associated to them. Figure 5.11 presents the cumulative distribution function for each mode of the "clean" track.

In a second group of 50 runs, we have replicated a real–life condition. The vision algorithm was, in this case, executed in a periodic task, receiving as input the whole data set (18400 frames per track), and processing one frame every $T = 100$ ms. The task was scheduled using SCHED_DEADLINE, with server period $T^s = 25$ ms and budget $Q^s = 4$ ms for the "clean" track.

The measured probability of respecting a deadline, expressed as the ratio between the number of jobs that finished before the deadline over the total number of jobs, was averaged through the 50 runs and compared with the one estimated by the MCTM model and the simplified i.i.d model presented in [84]. Note that the results labelled "i.i.d. approximation" assume that the computation times are described by an i.i.d. stochastic process even if they are not.

Figure 5.12 presents such a comparison, and shows two important things: first of all, the results obtained with MCTM are pretty similar to the ones obtained by executing a real application with SCHED_DEADLINE. Indeed, the experimentally obtained probability of respecting a deadline equal to the task period was 0.768; while the stochastic analysis performed assuming the MCTM indicated a probability of 0.765.

Then, the i.i.d. approximation results in an *overestimation* in the probability of respecting a deadline: 0.806 for the task period (a difference of around 4% for the task

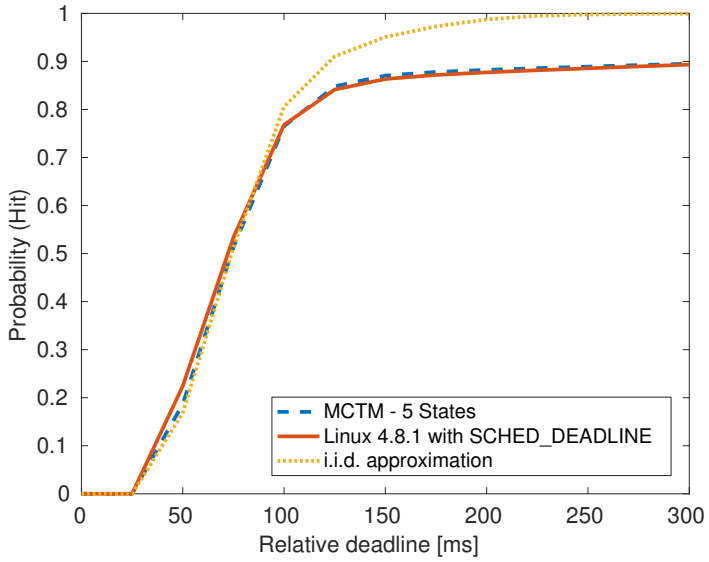

Figure 5.12: Comparison of the probability of respecting the deadline for different analysis methods, with a bandwidth fixed to 16% in the "clean" track. From 100 ms onwards, the i.i.d. approximation produces too optimistic results when compared with the experimental ones. By using the MCTM, the optimism is significantly reduced and the results from the analysis are in line with the experimental ones.

period but reaches 11% for 200 ms). Such an optimistic analysis can be dangerous when designing a real–time system. As shown, the MCTM allows to avoid this error.

Regarding the computation time needed by PROSIT to complete the analysis, when assuming i.i.d. computation times the time was 70.72 ± 13.14 ms; while the MCTM took 262.87 ± 6.23 ms to complete.

In the case of the "noisy" track, we found the same patterns presented in the case of the "clean" track. The computation times fluctuate in a recognisable trend, as shown in Figure 5.13 and the autocorrelation function reveals a strong correlation among the computation times.

The HMM identification algorithm was executed with the computation times obtained from the "noisy" track and presented in Figure 5.13. This allowed us to identify a 8–states MCTM. The validation algorithm successfully differentiated the 8 sub–sequences of independent computation times as detailed in Table 5.2 and Figure 5.14.

The results obtained from the HMM estimation procedure for the "noisy" track are compliant with the proposed MCTM, meaning that they correctly model the stochastic process of the computation times.

From the timing perspective, the estimation parameters were the same as in the case of the "clean" track, namely: the number of states ranged from 2 to 10, the number

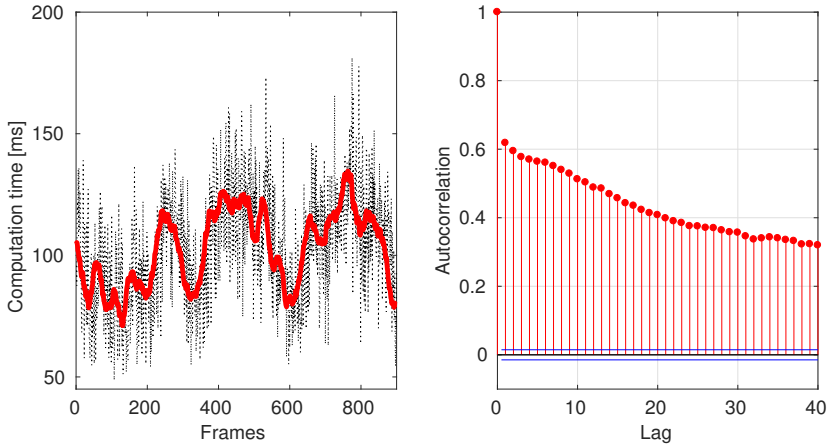

Figure 5.13: The left plot shows the computation times (dotted line) of the images taken by the robot while navigating the "noisy" environment along with a superimposed (solid line) moving average to highlight the trend. The right plot shows the autocorrelation function of those computation times for different "lags" or time instants.

of iterations was set to 100, and the number of samples was set to 18400. The time required by PROSIT to estimate the HMM parameters was 2479.15 $\pm$ 8.18 seconds for the identification and 389.71 $\pm$ 0.06 ms for the validation.

Regarding the probabilistic guarantees, the task executing the lane detection algorithm over the "noisy" track was scheduled as a periodic task with $T = 200$ ms, server period $T^s = 50$ ms and budget $Q^s = 35$ ms. Also in this case, as presented in Figure 5.15, we observe a good match between the proposed technique and the experimental result.

The experimentally obtained probability of respecting a deadline equal to the task period was 0.646; while the stochastic analysis performed assuming the MCTM indicated a probability of 0.642. In this case, the error introduced by making the i.i.d. approximation is evidently much larger: probability of 0.733, resulting in 9% of error but it reaches 23% for a deadline equal to 400 ms.

In this case, the computation time needed by PROSIT to complete the analysis when assuming i.i.d. computation times was 2.25 $\pm$ 0.05 seconds; while the MCTM took 1292.09 $\pm$ 0.84 seconds to complete.

In another set of experiments we compared the accuracy of the MCTM for different values of assigned bandwidth. Figure 5.16 shows the accuracy of the MCTM approach when the relative deadline is fixed to the task period ($T = 100$ ms), and different options of bandwidths are explored for the case of the "clean" track. Once again, the proposed approach shows a very good performance when compared with the real–life application. Similar results were obtained for the "noisy" track, as shown in Figure 5.17.

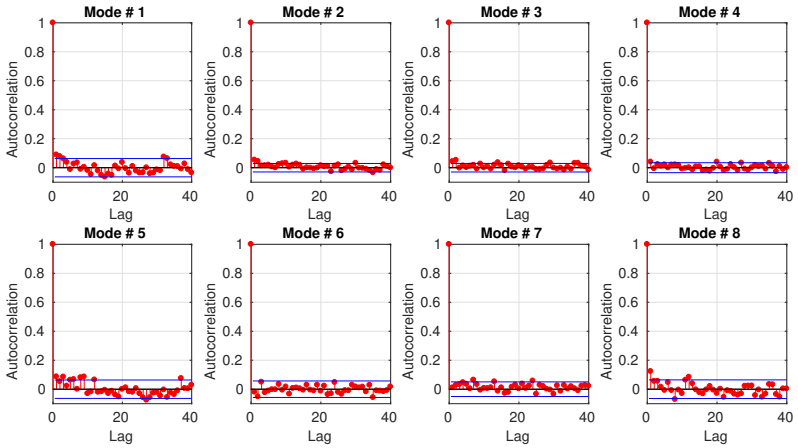

Figure 5.14: Autocorrelation function obtained from the 8 different modes in which the computation times presented in Figure 5.13 were classified. The strong correlation presented in the original sequence of computation times has disappeared. The 8 resulting sequences of computation times show a very low correlation.

The close match between the analysis and the experimental data is apparently the result of a good correspondence between the process expressing the computation time and the MCTM. For other applications, the MCTM fitting could not be so good or require a number of modes that is beyond the reach of our analysis tool.

5.5.2 The Reactive Planning Application

The second application correspond to the: *reactive planning*, described in Section 5.1.2, where a path planning algorithm is used to modify the original path when the robot encounters pedestrian while navigating its environment.

Simulation Setup

The computation times of the reactive planning algorithm have been measured by running 25 simulations on a realistic scenario. During each simulation, the FriWalk moves on a reference path for 600 seconds. While the walker moves along the reference trajectory, the relevant portion of the map around the robot is populated by a given number (between 0 and 5) of random moving humans.

For each human, three parameters are set: 1) a random initial position, placed between 3 and 6 meters ahead; 2) a random orientation; and 3) a random goal. At each step of the simulation, the moving agents in the field of view of the walker are determined, and the reactive planner algorithm is executed to update the current reference trajectory accordingly.

Table 5.2: Results of the numerical independence tests for the 8 sub–sequences estimated by the Baum–Welch algorithm. A p–value greater than 0.01 allows us to accept the independence assumption.

State	z–statistic	p–value
1	−1.5990	0.0549
2	−1.7215	0.0426
3	−1.8066	0.0354
4	−1.8218	0.0342
5	−1.8322	0.0335
6	0.4625	0.6781
7	2.0446	0.9796
8	−1.9505	0.0256

Whenever the walker overtakes all the moving agents in the current zone, a new set of random moving agents is generated to replace the previous one, and the simulation proceeds.

Experimental Results

The set of computation times obtained from the simulations was randomly divided in 3 sub–sets to perform the analysis and experimental validation of the proposed method. Specifically, the group of 25 simulations was classified into:

- A training set of 5 sequences, which was analysed using the proposed HMM identification techniques.
- A simulation set of 10 sequences. These sequences were used by the resource reservation simulator to estimate the probabilities of deadline miss.
- An i.i.d. approximation set of 10 sequences. From each sequence, we obtained a probability distribution of the computation times and perform the stochastic analysis over this distribution, assuming the sequence as if it was obtained from an i.i.d. process.

Figure 5.18, presents an excerpt of 1500 jobs from a training sequence. As expected, the computation time fluctuates but it has recognisable trends, revealed by the 25 samples moving average drawn in red and superimposed on the trace. Such trends are reflected into the autocorrelation function plotted on the right part of the figure. This correlation

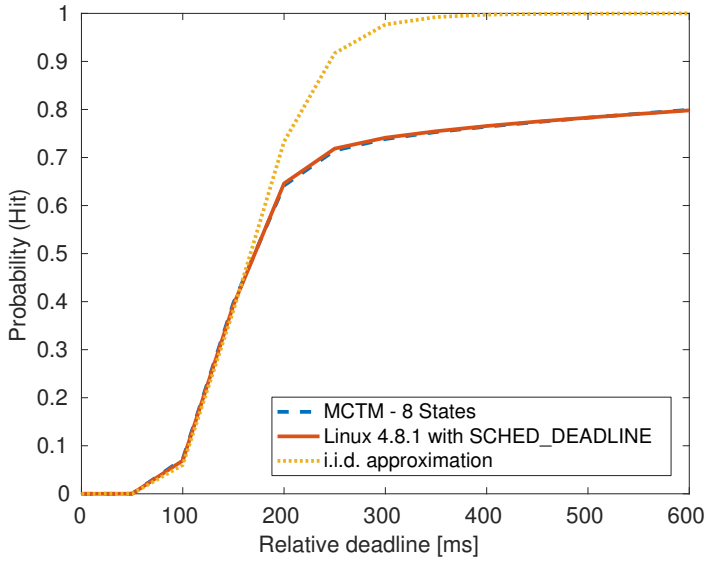

Figure 5.15: Comparison of the probability of respecting the deadline for different analysis methods, with a bandwidth fixed to 70% in the "noisy" track. From 175 ms onwards, the i.i.d. approximation produces too optimistic results when compared with the experimental ones. Using the MCTM eliminates the optimism in the analysis and the results are in line with the experimental ones.

prevents the use of a simple probability distribution function $C(c) = \mathbf{Pr}\{c_j = c\}$ to describe the computation times.

This characteristic of the computation times makes them a good candidate for the application of the proposed MCTM stochastic analysis. As a first step, it is necessary to identify the underlying model and verify whether it satisfy the requirements of the model. As mentioned before, the identification process was performed over the 5 sequences of the training set.

The Baum–Welch algorithm received as input the sequences from the training set. The process was repeated multiple times for different numbers of modes (ranging from 2 to 24). The algorithm was not able to identify a number of modes that satisfy the requirements; however, the best approximation was obtained with a 18–states HMM. Figure 5.19 presents the cumulative distribution function for each mode.

For the validation of the model, the Viterbi algorithm was used to identify the most likely sequence of hidden states corresponding to the sequence of observed computation times. It was possible to associate each computation time to one of the 18 hidden states of the approximated MCTM. This allowed us to partition the sequence of computation times in 18 sub–sequences, to perform a numerical test for independence [73] on such sub–sequences, and to compute the autocorrelation for each one of them.

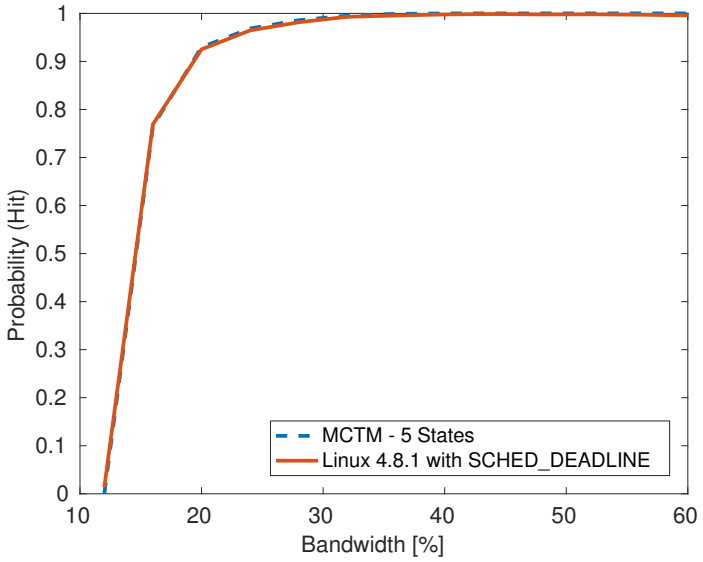

Figure 5.16: Distribution of the mean probability of respecting a deadline equal to the task period (100 ms) for different values of the bandwidth assigned to the task in the case of the "clean" track.

The results are presented in Table 5.3 and in Figure 5.20, and highlight a very interesting situation. Most of the operating modes, specifically 16 out of 18, passed the independence test; however, there are 2 operating modes (3^{rd} and 10^{th}) that did not pass the test.

Under this scenario, we cannot claim that the computation times can be described by the identified output probability distributions. Hence, the sequence of observed computation times cannot be modelled by a MCTM.

Nevertheless, despite the fact that the proposed mathematical model does not strictly apply, we decided to perform the stochastic analysis using the approximated MCTM previously identified. We were aware that the obtained results have to be considered as approximations. Still, a good quality of such approximation could make the case for the practical applicability of the proposed methodology.

In order to proceed with the stochastic analysis, the reactive planning algorithm was considered as a periodic task activated every $T = 200$ ms. The scheduling parameters were: server period $T^s = 40$ ms and budget $Q^s = 8$ ms. The simulation set was used to determine the empirical cumulative distribution function of the delays (probability of respecting the deadline).

The simulated probability of respecting a deadline, expressed as the ratio between the number of jobs that finished before the deadline over the total number of jobs, was compared with the one estimated by the approximated 18–states MCTM model and the

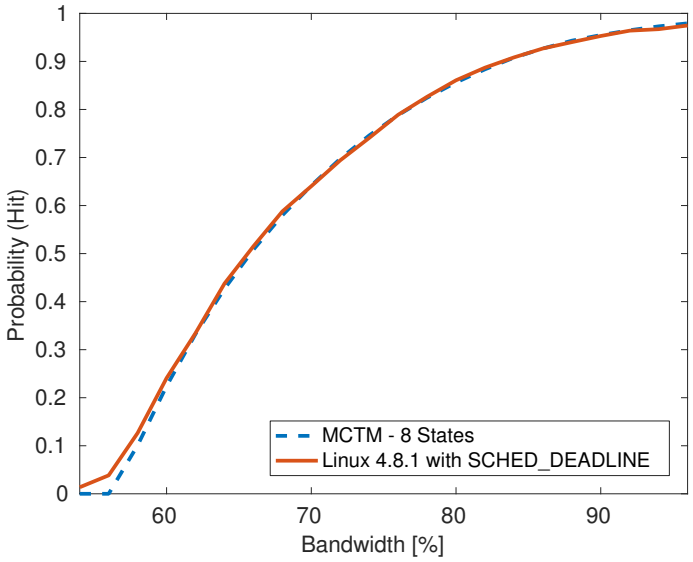

Figure 5.17: Distribution of the mean probability of respecting a deadline equal to the task period (200 ms) for different values of the bandwidth assigned to the task in the case of the "noisy" track.

simplified i.i.d model presented in [84].

Figure 5.21 presents such a comparison, and shows two important things: first of all, as observed in the case of the lane detection application, the i.i.d. approximation results in an *overestimation* in the probability of respecting a deadline: 0.794 for the task period (a difference of around 3% for the task period but reaches 10% for 400 ms).

Second, and more important, the results obtained with the approximated MCTM match the ones obtained by the simulator. Indeed, the simulated probability of respecting a deadline equal to the task period was 0.7611; while the stochastic analysis performed considering the approximated MCTM indicated a probability of 0.7612.

This result is very important for two reasons. 1) it suggests that there are certain applications which cannot be strictly modelled by a MCTM but they are suitable to be approximated by a MCTM. The use of this approximation produces tighter guarantees of respecting the deadline when compared with the more optimistic and traditional i.i.d. assumption; and 2) it raises the question about the possibility of finding another MCTM approximation, possibly with a fewer number of modes, that is able to provide similar probabilistic guarantees even when the mathematical assumptions do not hold.

In order to test this second situation, we decided to explore the concept of *statistical distance*, namely the distance between two statistical objects; for instance, two probability distributions. In particular, we chose the Kullback–Leibler (KL) divergence [65] as a metric of how one probability distribution diverges from another one.

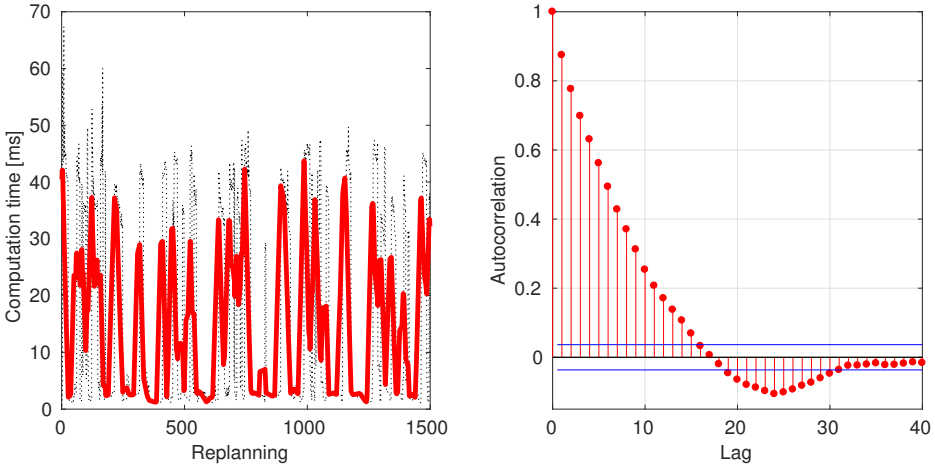

Figure 5.18: The left plot shows the computation times (dotted line) of the reactive planning process performed by the FriWalker while avoids moving obstacles along with a superimposed (solid line) moving average to highlight the trend. The right plot shows the autocorrelation function of those computation times for different "lags" or time instants.

We are interested in the number of modes for the approximated MCTM that produce the minimum distance between the probability distributions of the delay observed from the simulator and the ones estimated by the approximated N–states MCTM. By minimising the KL distance, we can find the probability distribution of the model that is closest to the probability distribution of the observations.

Table 5.4 reports the Kullback–Leibler divergence for a set of HMM estimations. The minimum distance correspond to the 9–states MCTM but there are several other options with a small distance that could be considered for the stochastic analysis. Additionally, Figure 5.22 presents the comparison of the probability of respecting the deadline for the two options of the approximated MCTM: the 18–states MCTM selected by the identification process and the 9–states MCTM chosen by the KL minimum distance along with the probabilities obtained from the simulator and the traditional i.i.d. approximation.

It is possible to observe that the probability of respecting the deadline between the approximated MCTM is very close: 0.761 for the 18–states MCTM and 0.756 for the 9–states MCTM. These results allows to conclude that the level of approximation that we have reported could be acceptable in several cases.

5.5.3 Scalability Issues

Another important factor that we wanted to analyse about the proposed solution was the scalability of the analysis when multiples operating modes are considered. It is easy to imagine that the tractability of the system as well as the time required to solve the

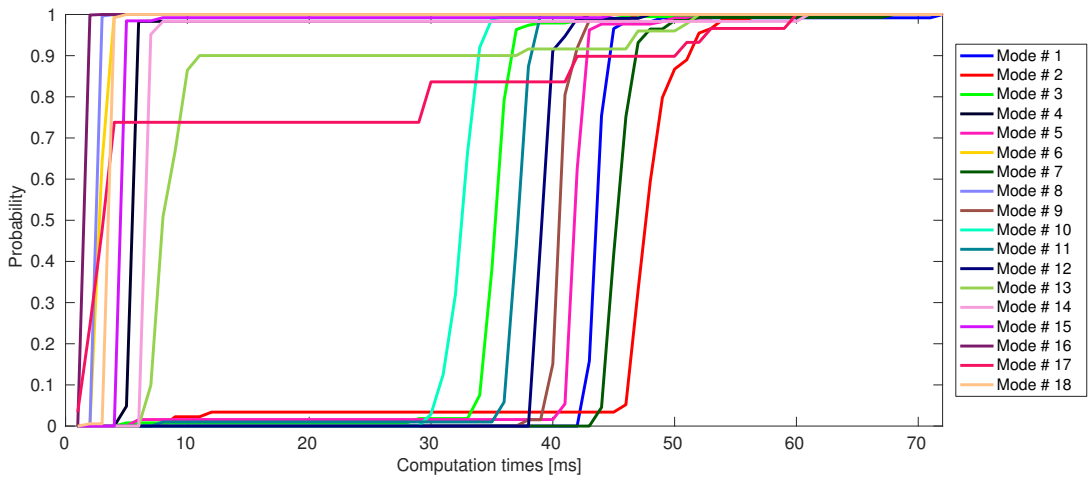

Figure 5.19: Empirical cumulative distribution functions for the 18 different operating modes. These modes were identified by applying the Baum–Welch algorithm to a sequence of computation times measured from the reactive planning application.

analysis are highly dependent on the number of operating modes.

When dealing with real–life applications, the underlying Markovian model is unknown. Therefore, to accomplish this set of experiments, we considered a set of 5 periodic task, identified from τ_1 to τ_5 with computation time described by a MCTM whose distributions were synthetically generated in simulation from known HMMs. The number of states for each MCTM was $N_i = \{2, 4, 8, 16, 32\}$ respectively.

The computation time for each mode of every task is an independent and identically distributed process generated according to a beta distribution with support $c \in [6000, 102000]$ μs. The parameters of the distribution were randomly obtained to comply with the desired distribution support.

From each known HMM, we generated 105 different sequences of computation times, where each sequence was composed of 15000 samples. These sequences were divided into:

- A training set of 5 sequences, which was analysed using the HMM identification techniques presented in Section 5.4.1 to identify the underlying MCTM that generated the sequence of computation times.

- A simulation set of 50 sequences. These sequences of computation times were used by the resource reservation simulator to measure the required time to compute the probabilities of deadline miss.

- An i.i.d. approximation set of 50 sequences. From each sequence, we obtained a probability distribution of the computation times and perform the stochastic analysis

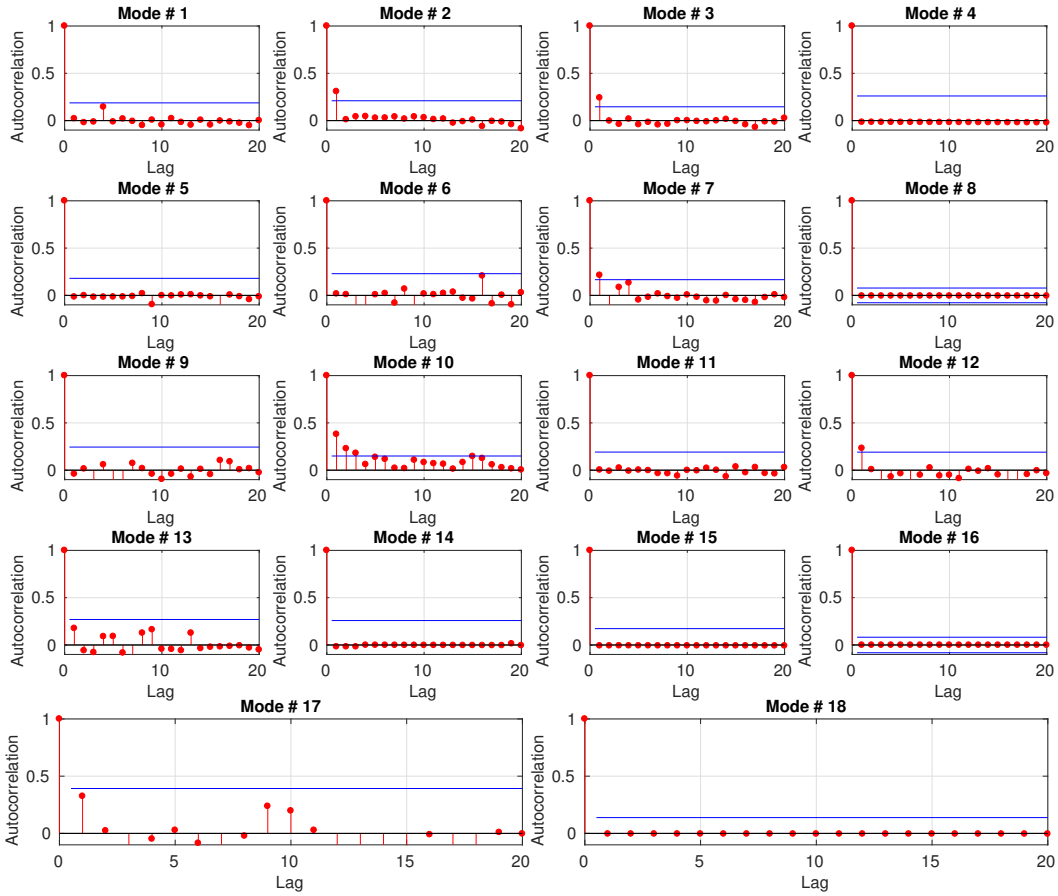

Figure 5.20: Autocorrelation function obtained from the 18 different modes in which the computation times presented in Figure 5.18 were classified. The strong correlation presented in the original sequence of computation times has been reduced. Most of the 18 resulting sequences of computation times show a very low correlation; however, some of the operating modes still show some correlation in their computation times.

over this distribution. Once again, we measured the required time to perform such analysis (assuming the sequence as if it was obtained from an i.i.d. process).

HMM Estimation

As a first step, we used the training set to estimate and validate the underlying MCTM. PROSIT was fed with the sequences of the set. The process was repeated 2 times, for different numbers of operating modes (ranging from 2 to N_i). In order to speed up the identification process, the generated samples were resampled with $\Delta = 1000\,\mu$s and the Baum–Welch algorithm was set to perform 25 iterations in the estimation process. The algorithm consistently identified N_i different modes, which presented independence in the

Table 5.3: Results of the numerical independence tests for the 18 sub–sequences estimated by the Baum–Welch algorithm. A p–value greater than 0.01 would allows us to accept the independence assumption; however, the 3^{rd} and 10^{th} sub–sequences did not pass the test.

State	z–statistic	p–value	State	z–statistic	p–value
1	0.3759	0.6465	**10**	**−6.0647**	**0.0000**
2	−1.8513	0.0321	11	−0.8001	0.2118
3	**−3.4817**	**0.0002**	12	−1.4848	0.0688
4	0.1873	0.5743	13	−0.9347	0.1750
5	0.593	0.7234	14	0.4452	0.6719
6	−0.2059	0.4184	15	0.2023	0.5802
7	−1.3303	0.0917	16	0.0581	0.5232
8	0.1673	0.5664	17	−1.4562	0.0727
9	0.4848	0.6861	18	0.0981	0.5391

computation times associated to them.

Table 5.5 shows the time required by PROSIT to identify the underlying MCTM. This time is the average of the 10 executions of the estimation process along with its corresponding 95% confidence interval. The identification process assumes no prior knowledge about the original number of states; hence, the estimation started with 2 states and kept increasing the number of states until the independence was found. The reported time considers the whole estimation process.

Stochastic Analysis

When the MCTM that properly describes the computation times was identified, the next step was to perform the stochastic analysis. In order to reason on the scalability of the system, we focused on 2 aspects: size of the generated matrices and total time of the analysis.

As mentioned before, the tractability of the system depends on the dimension of the probability transition matrix P of the QBDP presented in Equation 5.8 and described in Section 5.3.1.

The dimension of P depends on the scaling factor Δ presented in Section 3.4, on the number of modes identified for the MCTM, on the task period (T) and on the scheduling parameters (Q^s, T^s), as shown in Equation (5.8). It is easy to appreciate that, three of the factors maintain their values despite of the model for the computation time. Indeed, in order to make a fair comparison: the task period, the scheduling parameters and Δ

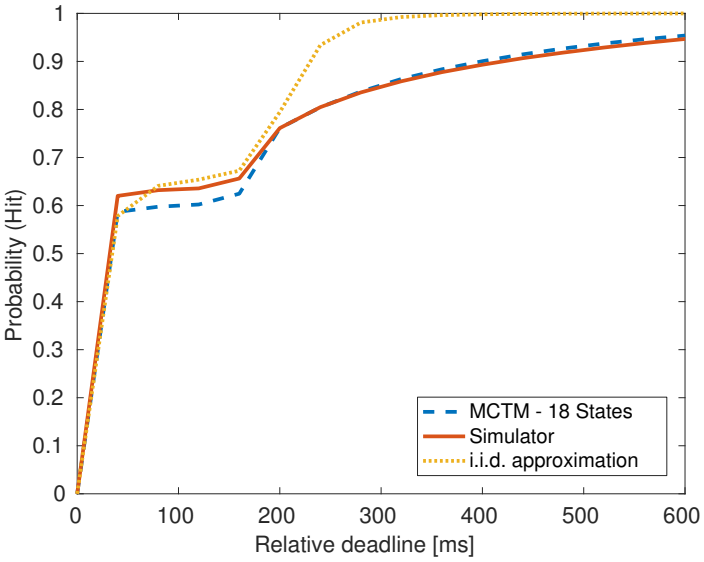

Figure 5.21: Comparison of the probability of respecting the deadline for different analysis methods, with a bandwidth fixed to 20% in the reactive planning application. From 200 ms onwards, the i.i.d. approximation produces too optimistic results when compared with the experimental ones. By using the MCTM, the optimism is significantly reduced and the results from the analysis are in line with the experimental ones.

must be equal for the analysis.

Therefore, the only remaining factor affecting the size of the matrix is the number of modes. Table 6.8 presents the dimension of the probability transition matrix for the different number of operating modes and for 2 possible values of Δ: 1000 μs and 250 μs respectively. Note that the size of the matrix for the MCTM increases proportionally to the number of modes with respect to the i.i.d. approximation matrix.

Besides affecting the size of the matrix, a large number of modes imply that the computation time required by PROSIT to perform the stochastic analysis is incremented. This computation time is presented in Figure 5.23. Note that the computation time grows really fast when the number of modes is incremented; for instance, consider a 8–states MCTM with $\Delta = 1000\,\mu$s, the average computation time required amounts to 15.80 ± 0.03 seconds. By setting $\Delta = 250\,\mu$s, the average time increases up to 1269.05 ± 72.21 seconds, namely two orders of magnitude greater. This behaviour is consistent for all the number of modes.

Table 5.4: Results of the Kullback–Leibler divergence metric. The minimum distance correspond to a 9–states MCTM, which represents the MCTM whose distribution of the delays is the closest to the distribution of the empirically obtained (measured) delays.

# of States	KL divergence	# of States	KL divergence
2	∞	14	0.012691
3	∞	15	0.020064
4	∞	16	0.018737
5	0.083273	17	0.016614
6	∞	18	0.014136
7	0.030806	19	0.022019
8	0.031249	20	0.015987
9	**0.011496**	21	0.016736
10	0.016214	22	0.016327
11	0.027950	23	0.013309
12	0.024869	24	0.016219
13	0.014049	25	0.018090

5.6 Applicability on Multi–Core Systems

As mentioned before, the proposed technique was developed to be applied on single–processor systems with the assumption that other types of interference between the tasks can be neglected. However, modern processors are multi–core where the interference between tasks could increase considerably; hence it is important to discuss how the proposed technique could be applied to these new processors.

When dealing with multi–processor architectures, there are two approaches for real–time scheduling: *global* and *partitioned* scheduling. In global scheduling, all ready tasks are put in the same data structure and then selected to execute on the different cores. In partitioned scheduling, instead, a list of ready tasks is maintained for each processor.

The proposed method can be applied with partitioned scheduling, ensuring that each task is associated with a specific *cpuset*, which is a logical entity provided by the Linux kernel consisting of CPU and memory nodes. A set of tasks can be assigned to a cpuset binding the resources that they use. If the cpuset contains one single CPU then assigning a set of tasks to the cpuset causes that the scheduler behaves as a truly partitioned scheme with a very small overhead.

By assigning the set of tasks to a specific CPU it is possible to carry out the stochastic analysis of the tasks as if they were executed on a single–processor system. It is important

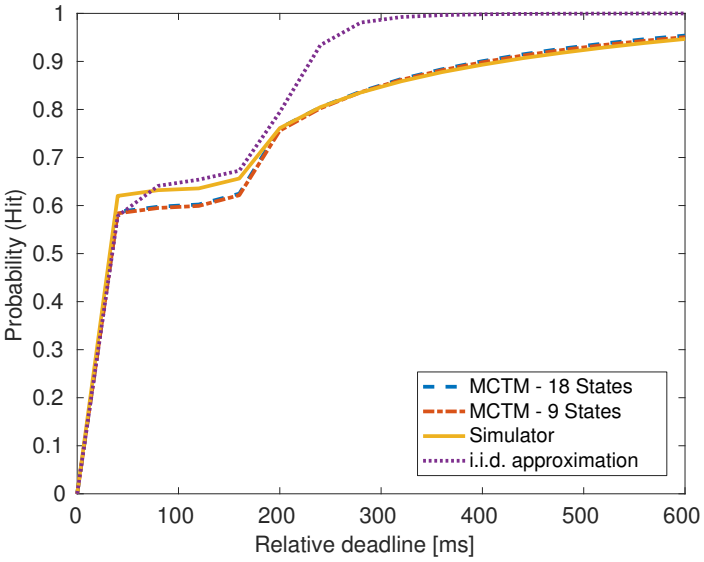

Figure 5.22: Comparison of the probability of respecting the deadline for different analysis methods, with a bandwidth fixed to 20% in the reactive planning application. This plot include, in dash–dotted red, the approximated 9–states MCTM recommended by the Kullback–Leibler divergence metric. Note that the recommended MCTM is slightly more conservative than the identified one.

to respect Equation (2.3), which indicates that the total assigned bandwidth cannot be greater than 1; in other words, the total CPU utilisation cannot exceed 100%.

This constraint leads to another possible application of the proposed analysis. Let us assume the presence of a group of soft real–time tasks running on a multi–core system. The tasks are required to satisfy a given Quality of Service (QoS) metric (e.g., minimum deadline hit probability), hence it is necessary not only to perform a stochastic analysis to estimate the deadline hit probability for each task but also to minimise the bandwidth assigned to each task while respecting the required QoS.

Since the tasks are running on a multi–core system, the next step is to devise an approach for the assignment of the different tasks to the cpusets in such a way that the CPU load remains balanced between them.

Table 5.5: Time required by PROSIT to perform the HMM identification: estimation and validation, for different number of states. This time correspond to the average of 10 trials over 15000 samples with 25 iterations of the Baum–Welch algorithm.

Task	# of States	Estimation [s]	Validation [ms]
τ_1	2	10.28 ± 1.58	15.13 ± 0.12
τ_2	4	65.83 ± 3.08	46.52 ± 0.33
τ_3	8	357.24 ± 21.27	113.35 ± 0.27
τ_4	16	3731.25 ± 102.91	273.73 ± 0.70
τ_5	32	45265.61 ± 1334.40	777.58 ± 5.12

Table 5.6: Comparison between the dimension of the probability transition matrix of the QBDP for the case of computation times described by: an i.i.d. process and a MCTM. Two different values of Δ are considered. The matrix of the MCTM is roughly equal to the i.i.d. matrix times the number of states.

Task	# of States	Matrix size			
		$\Delta = 1000$		$\Delta = 250$	
		IID	MCTM	IID	MCTM
τ_1	2	90×90	184×184	358×358	724×724
τ_2	4	106×106	448×448	422×422	1744×1744
τ_3	8	176×176	1456×1456	698×698	5776×5776
τ_4	16	160×160	2688×2688	630×630	10688×10688
τ_5	32	180×180	6144×6144	716×716	24384×24384

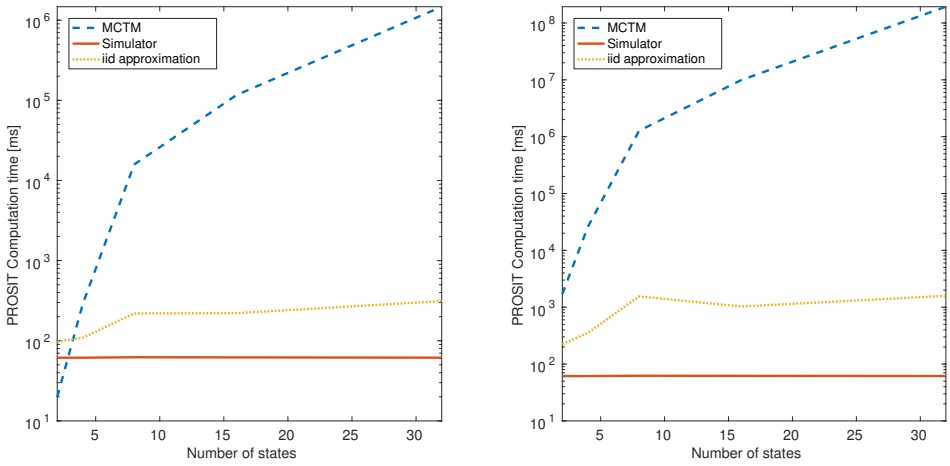

Figure 5.23: Time required by PROSIT to obtain the results of the stochastic analysis of real–time tasks for different number of states. The scaling factor Δ was set to 1000 μs for the left plot; while for the right plot it was set to 250 μs. Note that the choice of Δ is a key component for the analysis since there is an evident trade–off between: time required to perform the analysis and accuracy of the results.

Chapter 6

The PROSIT Tool

In this chapter, we present an extensible and open source design tool, called PROSIT[1], that facilitates the access to the probabilistic analysis of soft real–time systems for a potentially large number of researchers and industrial practitioners. PROSIT is implemented as a C++ library under the GNU GPL license and is available online including several examples and datasets.

The tool tackles two different problems in the stochastic analysis of soft real–time systems: 1) enables the probabilistic analysis of the temporal performance of a real–time task under fixed–priority and resource reservations scheduling algorithms; and 2) offers an automatic procedure for the synthesis of scheduling parameters for resource reservations that optimise a quality metric related to the probabilistic behaviour of the tasks.

These problems are henceforth referred to as: the Analysis and the Synthesis Problem. When the tool is used for analysis, it computes the response time distribution of the tasks, and hence, the probability of meeting the deadline. On the other hand, when the synthesis abilities of PROSIT are used, the tool finds an optimal allocation of the bandwidth between the different tasks in the system, along with the probability of meeting the deadline for each task.

The Analysis Problem can be shortly described as follows:

Problem 1 *Given a set of applications, each one characterised by: the probability distribution of its computation and inter–arrival times, a scheduling algorithm, a set of scheduling parameters, a sequence of h probabilistic deadlines, a solution algorithm for the probabilistic guarantees, and an optional quality function. Decide whether the steady state probabilities* $\left[\gamma_i^{(1)}, \ldots, \gamma_i^{(h)}\right]$ *are respected and compute the Quality* μ_i *for each of the applications.*

[1] https://bitbucket.org/luigipalopoli/prositool.git

In the Analysis Problem, the scheduling parameters are assumed to be chosen by the designer and are given by the priorities (P_i) for a fixed–priority scheduler and by the pair (Q_i^s, T_i^s) for a resource reservation scheduler. The computation time is, generally speaking, a MCTM described by a triplet $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$ as discussed in Section 5.2.

In the current version, the full model has been implemented only for resource reservation scheduling. For fixed–priority scheduling, to the best of our knowledge, there is no stochastic analysis that considers the computation time as described by a MCTM, hence PROSIT requires the tasks to be periodic and with i.i.d. computation time.

The Synthesis Problem is different since it requires the computation of optimal scheduling parameters. The idea is that a choice of scheduling parameters determines a set of steady state distributions, hence a different quality for each of the applications. The different qualities can be collected to form a global quality function $\mathcal{F}(\mu_1, \ldots, \mu_n)$.

The Synthesis Problem can be expressed as:

Problem 2 *Given a set of applications, each one characterised by: the probability distribution of its computation and inter–arrival times, the reservation period (T^s), a sequence of probabilistic deadlines, a solution algorithm for the probabilistic guarantees, a quality metric as a function of the probability of meeting the deadline, and constraints on the minimal value of the quality ($\underline{\mu}_i$). The quality of the different tasks can be combined into a global quality function. Maximise the global quality function $\mathcal{F}(\mu_1, \ldots, \mu_n)$ subject to $\mu_i \geq \underline{\mu}_i$ and to a schedulability condition.*

Currently, PROSIT solves the synthesis problem only for the case of CPU reservations, for which the schedulability condition can be set as $\sum_{i=1}^{n} B_i \leq 1$. In the case of fixed–priority scheduling algorithms, the scheduling parameter is the task priority (P_i). Hence, in order to solve the synthesis problem it is necessary to identify a priority assignment that respects a certain quality metric. Maxim et al. [77] presented three optimal priority assignment algorithms for tasks described by probabilistic computation times, where the quality metric is the maximum permitted deadline miss ratio. Solving the synthesis problem for fixed–priority scheduling implementing such algorithms will be considered in a future release of the tool.

6.1 PROSIT Overview

As mentioned before, the design of PROSIT rests on two pillars. The first one is the analysis of the stochastic behaviour of real–time tasks. As previously stated, much of the recent literature has reached the conclusion that such analysis can be conveniently reduced to the study of the steady state solution of a particular class of discrete–time Markov

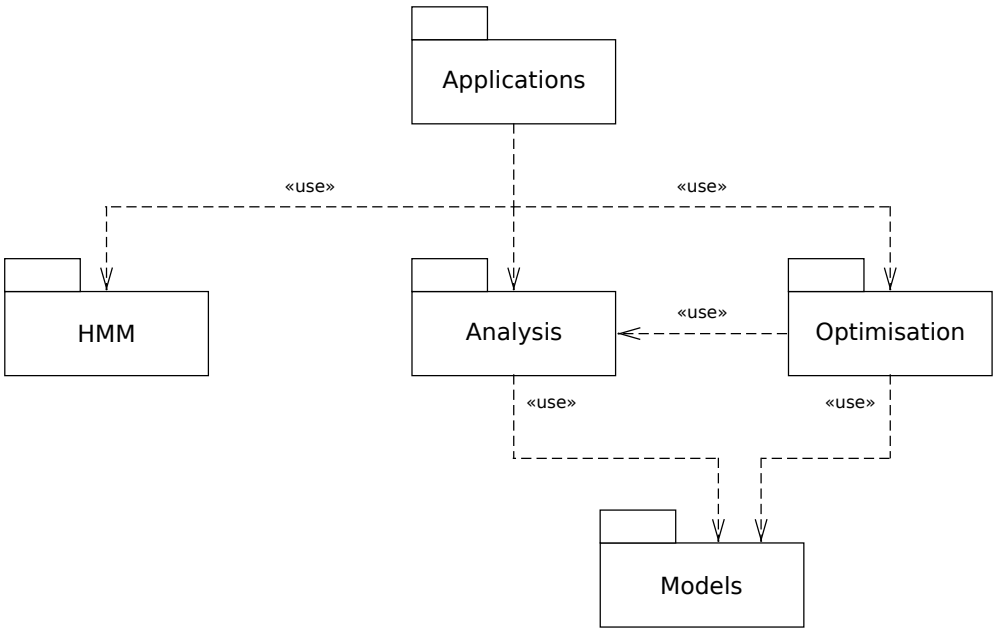

Figure 6.1: Package diagram for the PROSIT tool. This diagram details the main components of PROSIT and the relationship among them.

chains (DTMC), called Quasi–Birth–Death process (QBDP) [18]. This result holds both for fixed–priority scheduling [38] and for resource reservations [75], and it allowed us to use some of the most efficient numeric and analytic algorithms [19] as a basis for the implementation of the system analysis. Moreover, the resulting DTMC is a conservative approximation of the system, leading to a pessimistic result in the probabilistic analysis, as proved in [39, 84].

The second pillar is the definition of the Quality of Service as a function of the probability of meeting deadlines (or more generally, of the distribution of the delays). This allowed us to revisit, in a probabilistic framework, the ideas proposed in such frameworks as QRAM [91], and to develop algorithms for the optimal choice of the scheduling parameters. The solution of the optimisation problem is particularly efficient when we choose as a global quality metric the worst–case Quality between those experienced by the applications.

The PROSIT tool is a set of applications and libraries developed in C++, released under the GNU GPL license and designed to be flexible and extensible. It consists of a probabilistic analysis tool, a budget optimiser for the probabilistic analysis and an estimator of the parameters of the Markov Computation Time Model (MCTM) [6]. In order to perform all the vector and matrix calculations, PROSIT relies on the Eigen

library [52].

More specifically, Figure 6.1 shows a package diagram of PROSIT, where it is possible to identify its main components and their relations. Under the `Applications` package we group the set of programs that are directly accessible to the user, such as the probabilistic analysis solver or the budget optimiser.

The implementation of the parameter estimation of the MCTM can be found in the `HMM` package, while the `Analysis` package contains the different probability solvers, and the `Optimisation` package the classes and the functions related to the budget optimisation. The `Optimisation` package uses the `Analysis` package to iteratively perform a probabilistic analysis and solve the optimisation. In order to perform their tasks, both packages rely on the `Models` package, which groups all the classes characterising the tasks including the Quality of Service functions.

A closer look to the internals of the tool reveals the reduced class diagrams shown in Figure 6.2. This diagram describes the structure of the tool by showing the most relevant classes for probabilistic analysis and optimisation, along with their attributes, methods and the relationships.

In particular, there are two classes at the core of the PROSIT design: `TaskDescriptor` and `ProbabilitySolver`. The `ProbabilitySolver` class defines the different solvers implemented for the probabilistic analysis of real–time tasks. The solvers produce the steady state probability vector of the transition matrix of the QBDP, which is one of their attributes.

The `TaskDescriptor` class defines a task by its name, a map of probabilistic deadlines, the distribution of the inter–arrival time and the granularity (Δ) for resampling the computation time, as defined in Section 3.4. This class is specialised by two subclasses: `ResourceReservationTaskDescriptor` and `FixedPriorityTaskDescriptor`.

The former includes the scheduling parameters used for the resource reservation algorithm (e.g., budget and reservation period) and the distribution of the computation times modelled as a MCTM; namely, the number of modes, the transition probability matrix among the modes and the distribution of the computation times of each mode. Additionally, this class is associated with a `ResourceReservationProbabilitySolver`. Indeed, due to the temporal isolation property of the resource reservation scheduler, each task can be analysed independently by using a dedicated solver.

The `FixedPriorityTaskDescriptor` class defines the priority of the task, its initial offset and the distribution of the computation times modelled as an i.i.d. stochastic process. Contrary to the case of tasks scheduled by a resource reservation scheduler, the analysis of tasks scheduled by a fixed–priority scheduler has to be performed for the entire group of tasks sharing the processor. Such a group, specified through a vector of

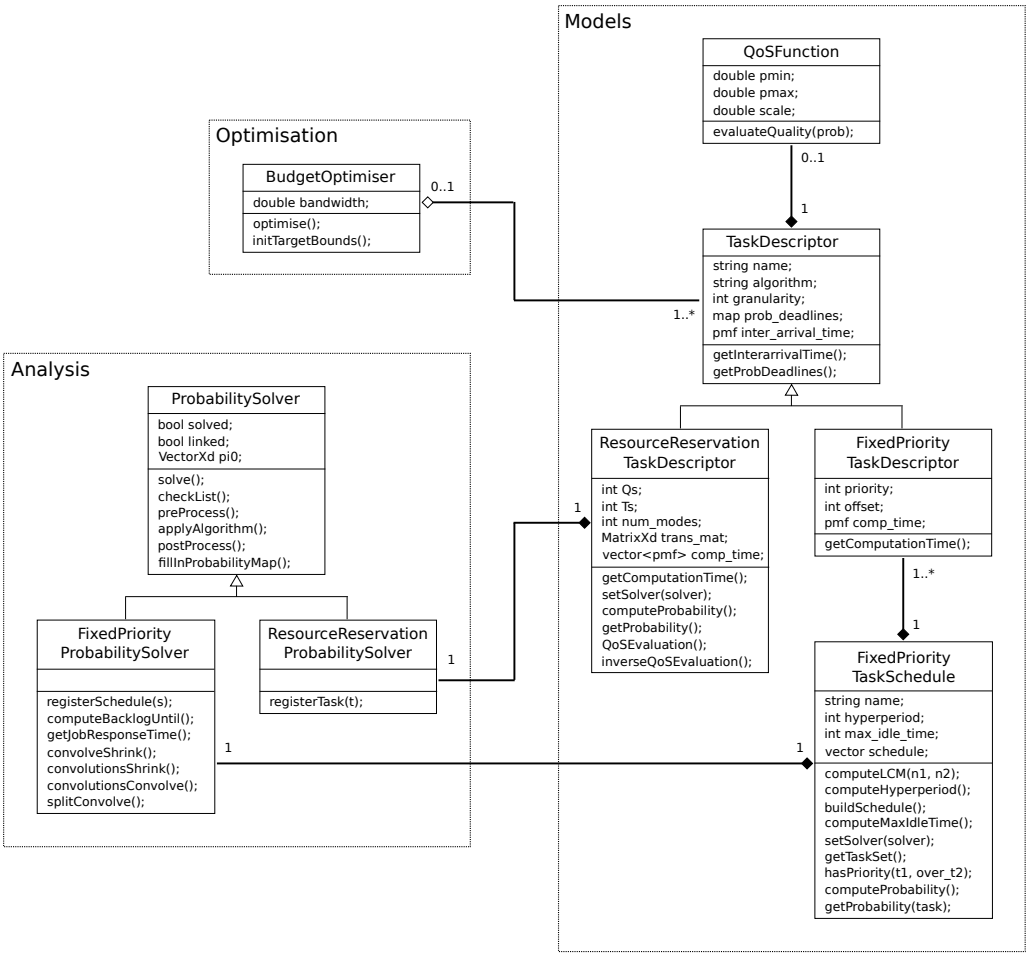

Figure 6.2: Minimal schematic of the class diagram for the PROSIT tool. This diagram reveals the structure: attributes, methods and relationships among the most important classes of PROSIT.

`FixedPriorityTaskDescriptors`, is contained in the `FixedPriorityTaskSchedule` class along with a `FixedPriorityProbabilitySolver`.

Finally, the `BudgetOptimiser` class is used to perform budget optimisation. This class contains a vector of `ResourceReservationTaskDescriptors` (associated with the tasks for which the budget optimisation is required) and with the available bandwidth, which has to be distributed among the tasks. Each task involved in the optimisation is associated with a `QoSFunction` to quantify its performance.

The typical workflow in using PROSIT is sketched in Figure 6.3, which represents two use cases related to the solution of the problems tackled by the tool. It is possible to observe that PROSIT accepts the information from the user through two XML system

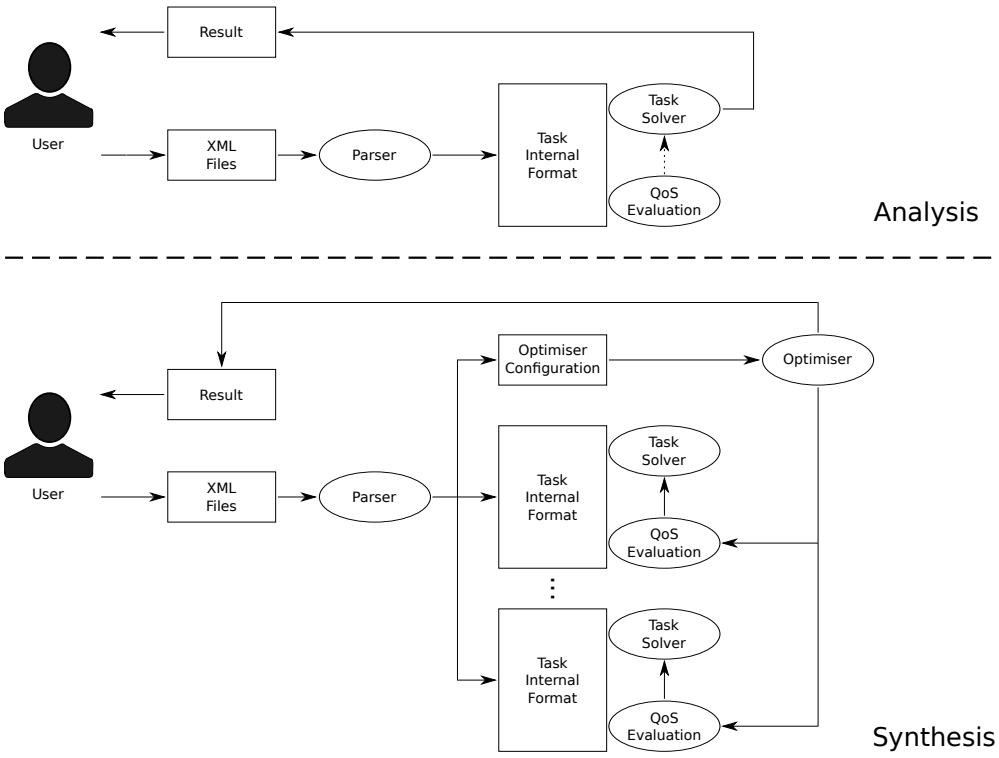

Figure 6.3: Use cases of the PROSIT tool when used for the analysis and the synthesis problem. The user provides the input by means of 2 configuration files. The analysis problem (top half) considers the tasks as if they are running "alone" in the CPU; while the synthesis problem (bottom half) optimises the allocation of the CPU, which is shared among a set of tasks.

specification files, which are internally parsed in order to create the instances of the objects needed for the probabilistic analysis or design.

As a result of the parsing process, the tool created the proper instances of the required objects (e.g., the task, the probability solver and the QoS function). It is important to point out that when solving the `analysis` problem, the tasks are assumed to execute "alone" in the system, hence the results obtained by the solver is presented to the user.

The situation is slightly different when solving the `synthesis` problem. In this case, there are several tasks sharing the CPU and the optimisation process iteratively perform a stochastic analysis on these tasks, adjusting the assigned budget at each iteration. When the synthesis requirements have been fulfilled, the results are presented to the user.

As a final and important observation, most of the features of the tool are exposed to the software developer through an API. This enables the use of the PROSIT library for applications other than the one proposed in the tool. For instance, one could use the

```
<taskset name="Set1">
  <task name="Tau1" type="periodic" algorithm="logarithmic">
    <period>300</period>
    <priority>2</priority>
    <computation_time1>examples/data/analysis/fp/example1/pmf_comp_time_task1.txt</computation_time1>
  </task>
  <task name="Tau2" type="periodic" algorithm="logarithmic">
    <period>400</period>
    <priority>1</priority>
    <computation_time1>examples/data/analysis/fp/example1/pmf_comp_time_task2.txt</computation_time1>
  </task>
</taskset>
<taskset name="Set2">
  <task name="Tau1" type="periodic" algorithm="cyclic">
    <period>100</task_period>
    <computation_time1>examples/data/synthesis/example1/pmf_comp_time_task1.txt</computation_time1>
    <qosfun type="linear">
      <pmin>0.01</pmin>
      <pmax>0.85</pmax>
      <scale>0.5</scale>
      <qos_min_target>0.1</qos_min_target>
    </qosfun>
  </task>
</taskset>
<taskset name="Set3">
  <task name="Tau2" type="periodic" algorithm="cyclic">
    <period>100</task_period>
    <num_modes>2</num_modes>
    <computation_time1>examples/data/synthesis/example1/pmf_comp_time_task2_mode1.txt</computation_time1>
    <computation_time2>examples/data/synthesis/example1/pmf_comp_time_task2_mode2.txt</computation_time2>
    <transition_matrix>examples/data/synthesis/example1/transition_matrix.txt</transition_matrix>
    <qosfun type="linear">
      <pmin>0.01</pmin>
      <pmax>0.95</pmax>
      <scale>0.5</scale>
      <qos_min_target>0.1</qos_min_target>
    </qosfun>
  </task>
</taskset>
```

Figure 6.4: Excerpt of an XML specification file for the description of the sets of tasks. Each task is composed of, at least, 1 task, which is described by a set of characteristics (e.g., name, solution algorithm, period, scheduling parameters, computation times, etc.). In the case of the analysis problem, the tag for the Quality of Service function is optional; while for the synthesis problem it is mandatory.

library for an admission test based on probabilistic deadlines.

6.1.1 XML Specification Files

As mentioned before, PROSIT receives the input from the user by means of two XML specification files. These files can be inserted by any XML editor (or by a GUI in a future development). One file contains a detailed description of the different tasks executing in the system, grouped in a number of task sets; while the other file defines the type of problem to be solved `analysis` or `synthesis` and associates each task set with the corresponding scheduling algorithm and its parameters.

As mentioned before, the tasks are described in an XML specification file, as shown in

Figure 6.4. The tasks can be grouped together to form a `taskset` which is characterised by a name `name` and it has to be composed of, at least, one task (`task`).

Each task is described by its symbolic name, a type, a solution algorithm for the probabilities, the initial phase, and the relative deadline. If fixed–priority scheduling is used, for each task we specify the priority (PROSIT assumes the convention for which the higher the number, the higher the priority).

In its current version, PROSIT implements 4 different solvers for the probabilistic guarantees: the Cyclic Reduction algorithm [19] (`cyclic`), the Logarithmic Reduction algorithm [66] (`logarithmic`), the Companion Form algorithm [84] (`companion`) and the approximated Analytic Bound [84] (`analytic`).

Additional elements are necessary to fill in the attributes of the task and must be part of the task XML entry. As an example, for periodic tasks it is necessary to specify the activation period, while for sporadic tasks it is necessary to specify the name of a file containing the distribution of the inter–arrival time (in the example, all tasks are assumed to be periodic). Note also that not all the attributes of the task are mandatory. For instance, if resource reservation scheduling is used, the `priority` is not required and can be safely omitted.

The computation time is also described by its Probability Mass Function (PMF), contained in an external file, if the stochastic process is assumed independent and identically distributed. In the case of MCTM computation time, the file must contain all the elements of the triplet $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$. The corresponding entry in the XML record are the number of modes (`num_modes`), the transition probability matrix (`transition_matrix`) and the distribution of the computation times of each mode (`computation_time#`).

In order to speed up the computation, it is also possible to specify the `granularity` parameter (the scaling factor Δ) used to resample the distributions. As mentioned in Section 3.4, a large value of the `granularity` reduces the time required for the computation of the steady state probability; however, the user has to pay the price of a coarser approximation for the computed probability.

An important problem is how to derive the distributions of the inter–arrival and of the computation time of the task (e.g., sequence of computation and inter–arrival times). This is relatively easy if the designer is in control of the source code and can instrument it with probes pinpointing the start and the termination of each job. The problem becomes much harder for legacy applications, which cannot easily be instrumented. Even in this case, reasonable estimates of the distributions can be derived by using two external modules: a tracer that operates inside the kernel and notes all the events related to the application and an event analyser, which detects periods and estimates the distribution of the computation time. More details on these external modules can be found in Section 6.3.

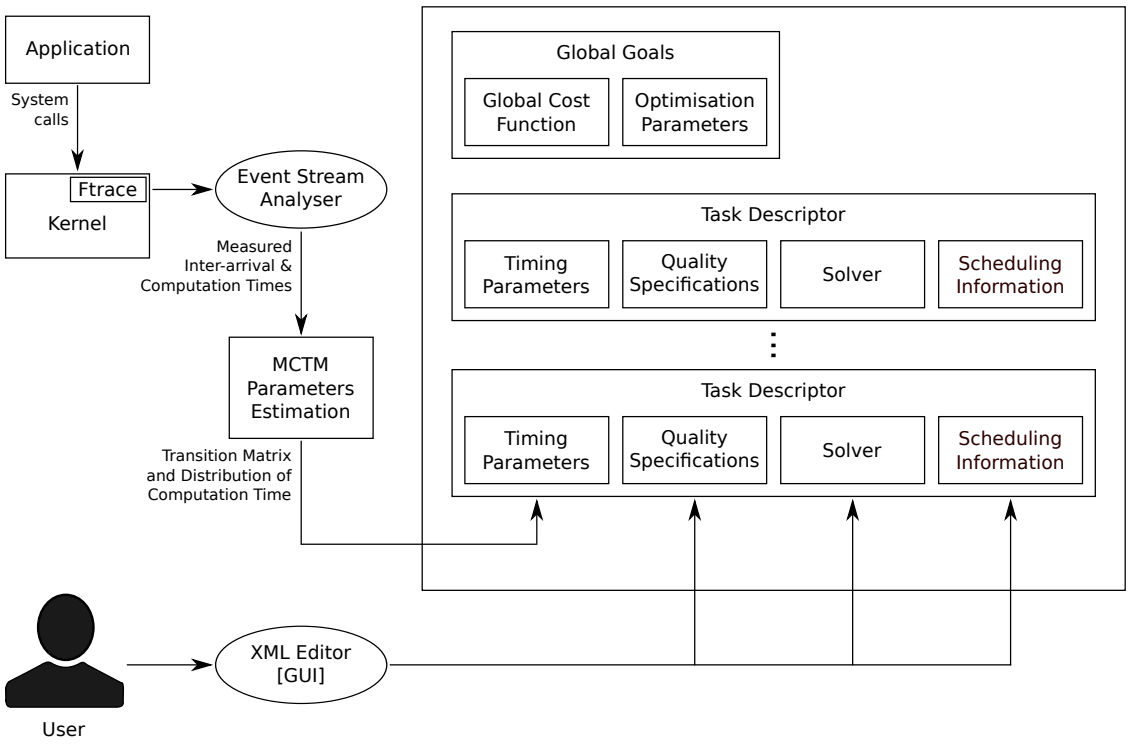

Figure 6.5: Definition of the synthesis problem with the PROSIT tool. The user provides the information to complete the XML specification files. The probability distribution of the computation times can be obtained empirically from the measured computation times. In the case of MCTM computation times, it is necessary to first estimate the parameters of the model before proceeding with the analysis.

Additionally, once the sequence of measured computation times has been obtained, it is necessary to estimate the parameters of the underlying MCTM. In order to be PROSIT compliant, the distributions of the inter–arrival and computation times have to be specified in a 2–column text file, where the first column represents the computation/inter–arrival time and the second column represents the probability. A possible workflow to derive the information required by PROSIT for its operations is shown in Figure 6.5.

As mentioned before, the second XML specification file, shown in Figure 6.6, defines the type of problem to be solved `analysis` or `synthesis` and associates the task set with the corresponding scheduling algorithm and its parameters.

The general structure of this XML file is similar in both cases. The first line defines the type of problem for which the tool is invoked: `analysis` or `synthesis`. In the specific case of a `synthesis` problem, it is necessary to specify the global quality function; for instance,

```
<analysis>
  <assignment>
    <taskset>Set1</taskset>
    <scheduler>
      <type>fixed-priority</type>
    </scheduler>
  </assignment>
  <assignment>
    <taskset>Set1</taskset>
    <scheduler>
      <type>resource-reservation</type>
      <reservation_period>100</reservation_period>
      <budget>60</budget>
    </scheduler>
  </assignment>
</analysis>


<synthesis method="infinity">
  <total_bandwidth>0.85</total_bandwidth>
  <assignment>
    <taskset>Set2</taskset>
    <scheduler>
      <type>resource-reservation</type>
      <reservation_period>25</reservation_period>
    </scheduler>
  </assignment>
  <assignment>
    <taskset>Set3</taskset>
    <scheduler>
      <type>resource-reservation</type>
      <reservation_period>20</reservation_period>
    </scheduler>
  </assignment>
</synthesis>
```

Figure 6.6: Excerpt of two XML specification files for the solution of the analysis problem (top half) and the synthesis problem (bottom half). The task set to be analysed is associated with a scheduler, which is characterised by its scheduling parameters. In the case of the solution problem, it is necessary to include the available bandwidth to be allocated among the tasks.

the `infinity` norm maximisation, and the optimisation parameters, for example, the total available bandwidth (`total_bandwidth`) to be distributed.

It is worth to point out that in the `analysis` problem of resource reservations, these tasks are assumed to execute "alone" in the CPU, meaning that the assigned bandwidth could reach 100% of the processor for each task. On the contrary, for the `synthesis` problem, these tasks are assumed to share the CPU, meaning that the sum of the assigned bandwidth to each task cannot exceed the total available bandwidth.

6.1.2 Analysis

The current version of the tool restrict the analysis to periodic tasks with i.i.d. computation time for fixed–priority scheduling, while for resource reservations, it is possible to specify stochastic (i.i.d.) inter–arrival time and MCTM computation time. An option left to the user is to insert, in the corresponding XML file, the definition of the QoS function

for the tasks.

The analysis of the system requires to specify the scheduling parameters (the budget Q^s and the reservation period T^s for resource reservation scheduling). In our example in Figure 6.6, we consider the task set to be scheduled by a fixed–priority scheduler (first assignment) and by a resource reservation scheduler with scheduling parameters $T^s = 100$ and $Q^s = 60$ (second assignment).

As a result of the tool invocation, PROSIT loads and parses the XML specification files, collecting all the relevant information of the tasks and instantiates the C++ objects from the classes as shown in Figure 6.2. Clearly, the selection of the generated objects depends on the scheduling algorithm and on the solver selected by the user.

In the particular case of tasks scheduled using a resource reservation scheduler, a `ResourceReservationTaskDescriptor` object is created for each task. As mentioned before, this object has a corresponding `ResourceReservationProbabilitySolver` object that must be created and set through a call to the `setSolver(solver)` method. Additionally, if the user specifies an XML entry for the optional quality function, a `QoSFunction` object will be generated and linked to the task descriptor.

The most important method of the `ResourceReservationTaskDescriptor` object is the `computeProbability()` method. This method is in charge of computing the probability of respecting the deadline for the task by calling the `solve()` method from its associated `ResourceReservationProbabilitySolver`. This is a virtual method, which is overloaded with the specific solution strategy chosen by the user. If required, the quality corresponding to the probability of respecting the deadline is computed by the `evaluateQuality(prob)` method, which is itself a pure virtual function to allow for different quality models.

In the case of fixed–priority scheduling, the analysis is performed on a set of tasks, hence a `FixedPriorityTaskSchedule` object is created. This object handles a vector of `FixedPriorityTaskDescriptors`. In the same way, the `FixedPriorityTaskSchedule` object has a corresponding `FixedPriorityProbabilitySolver` object that must be created and set by calling the `setSolver(solver)` method.

After setting the `FixedPriorityProbabilitySolver`, it is possible to call the corresponding `computeProbability()` method from the `FixedPriorityTaskSchedule` in order to compute the probability of respecting the deadline for the task set.

As a result of the program execution, the tool prints out on the screen the probability of respecting the deadline (p_i), as shown in Figure 6.7. Additionally, the printout also includes the total computation time spent by PROSIT in the probabilistic analysis.

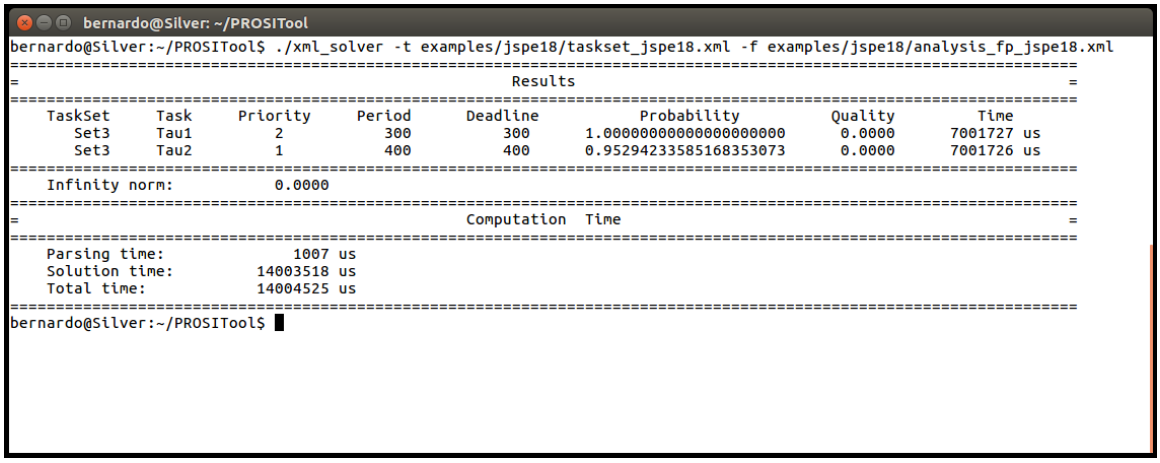

Figure 6.7: Screen capture of the output produced by PROSIT when queried for an analysis problem. The output includes some parameters of the task, the probability of respecting the deadline, the metric, if any, for the Quality of Service, and the time required to analyse each task and the whole process.

6.1.3 Synthesis

The solution of the synthesis problem is currently available only for the resource reservations. In this case, it is mandatory to define a Quality function for each task belonging to the task set. In the example presented in Figure 6.4, the user–provided Quality function is specified by a type (`linear`), meaning that there is a linear relation between Quality of Service and probability of meeting the deadline (see Section 6.4 for more details), and by a set of parameters. In the same entry, it is necessary to define the lower bound for the Quality of Service (`qos_min_target`).

It is noteworthy to point out that in the synthesis problem, the user does not specify the budget Q^s, which is computed by the optimisation algorithm. One could legitimately argue over the choice of fixing T^s and leaving Q^s as decision variable. The motivation is rooted in the philosophy of the resource reservations, in which T^s is used to control the granularity of resource allocation, while Q^s is used to control the bandwidth.

In the corresponding XML specification file (see Figure 6.6), the user also specifies the global quality function $\mathcal{F}(\mu_1, \ldots, \mu_n)$. This function composes the quality associated with each application, which is a non–negative real number. Possible choices are: the *infinity norm*, where $\mathcal{F}(\mu_1, \ldots, \mu_n) = \max_{i=1}^{n} \mu_i$; or, the *one norm*, where $\mathcal{F}(\mu_1, \ldots, \mu_n) = \sum_{i=1}^{n} \mu_i$.

After the tool execution is started, the parser generates a C++ instance for each task (`ResourceReservationTaskDescriptor`) in the task set and hands them over to the `GenericBudgetOptimiser` object, which calls the `optimise()` method.

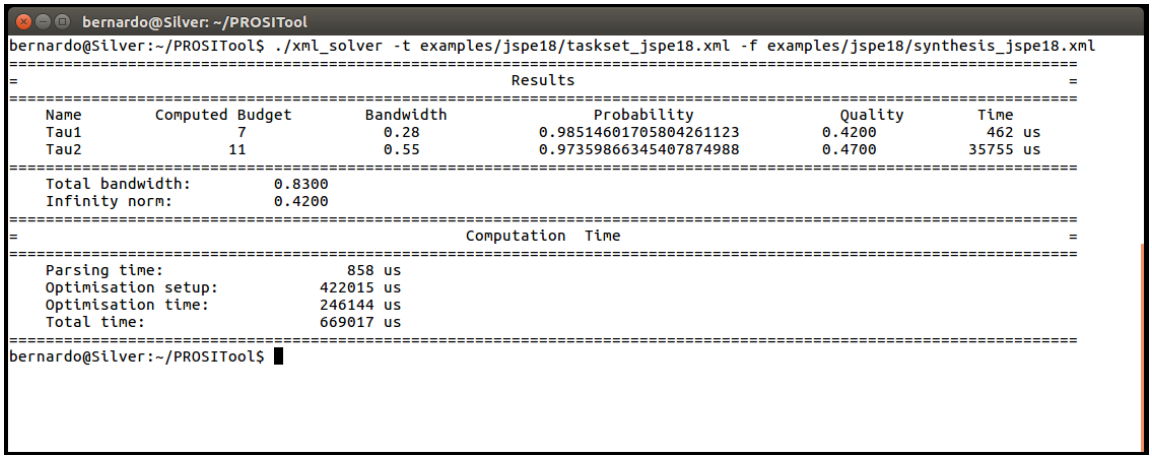

Figure 6.8: Screen capture of the output produced by PROSIT when queried for a synthesis problem. The output includes the name of the task, optimal bandwidth to be assigned in order to satisfy the required Quality of Service, the probability of respecting the deadline, the metric for the Quality of Service, and the time required to analyse each task and the whole process.

The optimisation algorithm iteratively calls the `evaluateQuality(prob)` method of the `QoSFunction` associated to the task, and indirectly the `solve()` method of the `ResourceReservationProbabilitySolver`, to compute the quality associated to a choice of decision variables or to estimate the gradient and eventually produces the optimal choice of the scheduling parameter (Q^s).

When the execution of the program has finished, the tool prints out on the screen the obtained results for each task (see Figure 6.8) including the assigned budget (Q^s), the probability of respecting the deadline (p_i) and the quality (μ_i). Additionally, the printout also contains information on the total computation time (along with its split between the different phases of the algorithm).

6.1.4 HMM Identification

As mentioned before, the implementation of PROSIT assumes that the computation time of the tasks are described by a Markov Computation Time Model (MCTM). In order to apply the stochastic analysis of a task whose computation time is described by a MCTM, it is necessary to express this model in terms of a Markov chain, identifying the transitions between the operating modes and the distribution of computation time in each mode, as described in Section 5.4.

This identification is performed by applying the theory and the techniques developed for hidden Markov models (HMMs) [89] over a long enough series of observed computation times. In [6], we have introduced an effective procedure for the identification of

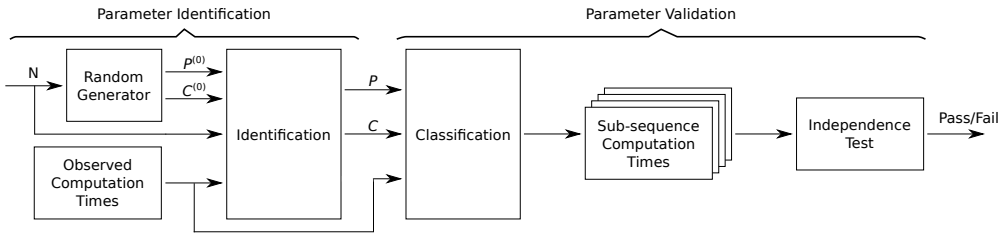

Figure 6.9: Workflow for the estimation of the MCTM parameters, where N is the number of operating modes, P is the transition matrix of the Markov chain and C is the distribution of the computation times. The identification process receives a sequence of measured computation times, the number of modes and initial random guesses for P and C. After the required number of iterations, it produces the estimations. The computation times are classified into N sub–sequences and then it is performed the numerical independence test over those sequences.

the parameters of a MCTM. These techniques, sketched in Figure 6.9, have been also implemented in PROSIT.

In particular, in order to identify the parameters of the HMM that best describe a given sequence of observed computation times, PROSIT implements the well–known Baum–Welch algorithm [12], which basically finds the set of parameters that maximise the likelihood that the given sequence of values is generated by the identified HMM.

Currently, this functionality is presented as a standalone program named `mctm_learner`. It receives as input: 1) a text file, in a 1–column fashion, containing the observed computation times, 2) the number of operating modes, 3) the number of random trials to choose the best solution, and 4) the granularity for resampling the observations (the scaling factor Δ).

Once the state transition matrix and the distribution of computation time in each mode have been estimated, it is important to check whether the identified MCTM correctly describes the computation times of the application. This verification is implemented in PROSIT by using the Viterbi algorithm [48], which generates the most likely sequence of internal states for the identified MCTM given the computation times sequence.

Additionally, the initial sequence of observed computation times is divided into different sub–sequences, one for each operating state, and an independence test over such sub–sequences must be performed in order to guarantee that the computation time in each state is independent and identically distributed.

Once again, a standalone program named `mctm_decoder` is available. This program accepts as input: 1) a text file, in a 1–column fashion, containing the observed computation times, 2) the number of operating modes, 3) the granularity for resampling the

observations (the scaling factor Δ), 4) the transition matrix between the operating modes, 5) the distribution of computation time in each operating mode, and 6) the significance level for the p–value.

The `mctm_decoder` program presents a table with the z–score and its corresponding p–value for each state in the Markov chain. Additionally, it shows whether the independence test was passed or not. Moreover, the program will create different text files containing the observations classified by state.

6.1.5 A Simulator for Resource Reservations

PROSIT also includes a simulator of a resource reservation scheduler. The simulator, a standalone program named `cbs_simulator`, accepts all the characteristic parameters of a real–time task scheduled with a resource reservation scheduler: task period (T), reservation period (T^s), budget (Q^s), and timing requirements (e.g., distribution of the inter–arrival and computation times). Additionally, it is necessary to indicate the length of the simulation, namely the number of jobs to generate.

The computation times can be described either by an independent and identically distributed stochastic process or by a MCTM. In the latter, the simulator also accepts the probability transition matrix of the Markov chain and generates a sequence of computation times obtained from the evolution of the corresponding Markov chain.

The simulator includes two possibilities for the computation times of the tasks: they are either described by a probability distribution or directly by a sequence of values. In the former, the simulator randomly generates a computation time from the distribution; while, in the latter, the times are extracted, one by one, from the sequence.

After the execution, the simulator produces two result files with: 1) the probability of respecting the deadlines, expressed as the ratio between the number of jobs that finished before the deadline over the total number of jobs, and 2) the timing details of each job: the activation, computation, finishing and response time.

6.2 Algorithms: Analysis and Synthesis

The cornerstone of PROSIT is the probabilistic analysis of soft real–time systems, implemented for both: resource reservations and fixed–priority scheduling. The stochastic analysis for CPU reservations is presented in Chapter 5; however, for completeness, we briefly present the analysis of a set of tasks scheduled by a fixed–priority scheduler. Additionally, it is presented the optimisation algorithm that solves the synthesis problem.

6.2.1 Probabilistic Analysis for Fixed–Priority Scheduling

The probabilistic analysis of a set of periodic tasks scheduled by a fixed–priority scheduler can also be modelled as a Quasi–Birth–Death Process, as described in [38]. The analysis computes the response time distribution of each task in the system, and, by extension, determines the probability of respecting the deadline for each task in the task set. The interested reader is referred to the original paper as we will briefly introduce only the most important concepts.

Recall that each job $J_{i,j}$ is characterised by its activation time $a_{i,j}$, its finishing time $f_{i,j}$, and its computation time $c_{i,j}$. The response time of the job is given by $r_{i,j} = f_{i,j} - r_{i,j}$. The response time is assumed to be a stochastic process $\mathcal{R}_i$ with PMF $\mathcal{R}_i(r) = \mathbf{Pr}\{r_{i,j} = r\}$.

The analysis relies on the stochastic regularity observed at the hyperperiod level. Hence, the probability distribution of the response time of task τ_i is represented by the average of the probability distribution of the response time of all the jobs from the task in a hyperperiod, as follows:

$$\mathbf{Pr}\{r_i = r\} = \frac{1}{m_i} \sum_{j=1}^{m_i} \mathcal{R}_i(r), \tag{6.1}$$

where $m_i = \mathcal{T}/T_i$ is the number of jobs from τ_i released in a hyperperiod of length $\mathcal{T}$.

To compute those response time distributions, the authors have defined the *P–level backlog observed at time t* ($\mathcal{W}_t^P$, with PMF $\mathcal{W}_t^P(w) = \mathbf{Pr}\left\{\mathcal{W}_t^P = w\right\}$), as the sum of the remaining computation times of all the jobs that have priorities higher than or equal to P at time t and that still have to be served.

If the P–level backlogs at time $t = 0$ (beginning of the hyperperiod) are known, then the P–level backlogs for $t \neq 0$ can be computed based on them. Hence, the analysis provides a way to compute a stationary distribution for the P–level backlog observed at the beginning of each hyperperiod k, denoted by $\mathcal{B}_k^P = \mathcal{W}_{t\mathcal{T}}^P$, with PMF $\mathcal{B}_k^P(b) = \mathbf{Pr}\left\{\mathcal{B}_k^P = b\right\}$. In particular, the analysis of Diaz et al. [38] proved that the stochastic process defined by the sequence: $\mathcal{B}_1^P$, $\mathcal{B}_2^P$, ..., $\mathcal{B}_k^P$, ... is a Markov chain whose transition matrix presents the same recursive structure as presented in Equation (3.8), where each column j represents the probability distribution of the backlog at the end of the first hyperperiod, if the initial backlog was j.

Once again, casting the system into the QBDP framework allows us to compute the steady state distribution of the P–level backlog, $\mathcal{B}_k^P(b)$, at the beginning of the hyperperiod (at time k). This steady state distribution allows us to compute the steady state distribution of the P–level backlog at any time k', with $k' > k$. Usually, the time k' is

the release time of the job and this backlog is called the steady state *job–level backlog*, $\mathcal{W}^{P}_{a_{i,j}}(w)$.

When the probability distribution of the job–level backlog for all the jobs in the hyper-period is computed, it is possible to compute the probability distribution of the response time of a job. For each job $J_{i,j}$, the response time can be computed as:

$$r_{i,j} = \mathcal{W}^{P_i}_{a_{i,j}} + c_{i,j} + \mathcal{I}_{i,j}, \tag{6.2}$$

where $\mathcal{W}^{P_i}_{a_{i,j}}$ represents the sum of the remaining computation times of all the jobs that have priorities higher than or equal to P_i (P_i being the priority of the task τ_i to which the job belongs) and are not completed up to the release time, and $\mathcal{I}_{i,j}$ is the interference on $J_{i,j}$ of all the jobs of higher priority than job $J_{i,j}$, released after job $J_{i,j}$.

Hence, the steady state probability distribution of the response time of job $J_{i,j}$, denoted as $\mathcal{R}_i(r)$, can be computed from the steady state probability distribution of the job–level backlog, $\mathcal{W}^{P}_{a_{i,j}}(w)$; the probability distribution of the computation time of the job, $\mathcal{C}_i(c)$; and the probability distribution of the computation times of the jobs released after time $a_{i,j}$ with a priority higher than that of job $J_{i,j}$. Note that this process is obtained from Equation (6.2), where the sum of independent random variables is computed by performing the convolution over its probability distributions.

Finally, when the probability distribution of the response time for each job of the task τ_i is obtained, it is possible to compute the probability distribution of the response time of the task τ_i by averaging the probability distribution of each job, as presented in Equation (6.1).

6.2.2 The Optimization Algorithm

As discussed before, the synthesis problem amounts to the solution of an optimisation algorithm, which in the current version of PROSIT is possible only for resource reservations. The solution is very efficient when the infinity norm is used and the following assumption can be made: *the quality increases if the budget reserved to the application, and hence the probability of meeting the deadline, increases.* In all the cases of practical relevance that we have examined, this assumption is easily verified.

The monotonicity of the function allows us to apply the efficient solution algorithm reported in Algorithm 1. The first lines of the algorithm (3 through 5) are to verify if the total bandwidth required to attain the lower bounds of the specification exceeds 100%, the problem being unfeasible in this case.

The search for the optimal solution is reduced to within two bounds (lines 6-7); the lower one ($\underline{\mu}$) derives form the lower bound constraints, while the higher one is obtained by assigning 100% of the bandwidth to the task ($Q^s = T^s$). The method

Algorithm 1 Solve Optimisation

Require: taskList; /* List of the tasks */; $\underline{\mu}_i$; /* Lower bounds for Quality */

Ensure: μ; /* Optimal value */; Q_i^s; /* Optimal budgets */

1: $B_A = 1.0$;
2: $\forall_i \ \overline{\mu}_i = \tau_i$.reservationPeriod();
3: **if** $\sum_{\tau_i \in \text{taskList}} \frac{\tau_i.\text{inverseQoSEvaluation}(\underline{\mu}_i)}{\tau_i.\text{reservationPeriod}()} > 1.0$ **then**
4: **return** UNFEASIBLE;
5: **end if**
6: $\overline{\mu} = \min_{i=1,\ldots,n} \overline{\mu}_i$;
7: $\underline{\mu} = \min_{i=1,\ldots,n} \underline{\mu}_i$;
8: **for** $\tau_i \in$ taskList **do**
9: $Q_i^s = \tau_i$.inverseQoSEvaluation($\overline{\mu}$);
10: **if** $Q_i^s < \tau_i$.inverseQoSEvaluation($\underline{\mu}_i$) **then**
11: $Q_i^s = \tau_i$.inverseQoSEvaluation($\underline{\mu}_i$);
12: taskList = taskList $\setminus \tau_i$;
13: $B_A - = \frac{Q_i^s}{\tau_i.\text{reservationPeriod}()}$;
14: **end if**
15: **end for**
16: **if** $\sum_{\tau_i \in \text{taskList}} \frac{Q_i^s}{\tau_i.\text{reservationPeriod}()} < 1.0$ **then**
17: $\mu = \overline{\mu}$;
18: **else**
19: **while** $\overline{\mu} - \underline{\mu} > 0$ **do**
20: $\mu = \left(\overline{\mu} + \underline{\mu}\right)/2$;
21: **for** $\tau_i \in$ taskList **do**
22: $Q_i^s = \tau_i$.inverseQoSEvaluation(μ);
23: **if** $Q_i^s < \tau_i$.inverseQoSEvaluation($\underline{\mu}_i$) **then**
24: $Q_i^s = \tau_i$.inverseQoSEvaluation($\underline{\mu}_i$);
25: taskList = taskList $\setminus \tau_i$;
26: $B_A - = \frac{Q_i^s}{\tau_i.\text{reservationPeriod}()}$;
27: **end if**
28: **end for**
29: **if** $\sum_{\tau_i \in \text{taskList}} \frac{Q_i^s}{\tau_i.\text{reservationPeriod}()} < B_A$ **then**
30: $\underline{\mu} = \mu$
31: **else**
32: $\overline{\mu} = \mu$
33: **end if**
34: **end while**
35: **end if**

`inverseQoSEvaluation()` computes the budget required to attain a specified level of quality; since the Quality is assumed monotone increasing, the inversion can be carried out by a simple dichotomic search.

This operation can be expensive because it entails repeated calls to the `solve()` method. The code segment between line 8 and 15 computes the budget required to each task for the upper bound $\overline{\mu}$. If some of the tasks is constrained to a lower bound $\underline{\mu}_i$ higher than $\overline{\mu}$, this task is removed from the subsequent search phases and it is allocated a bandwidth sufficient to attain its lower bound.

The execution of the algorithm is terminated if the total required bandwidth is lower than 100%, meaning that $\overline{\mu}$ is attainable and is therefore the optimum. In the opposite case, a binary search is carried out, in which the same steps applied to $\overline{\mu}$ between line 7 and 15 are applied to the midpoint between $\underline{\mu}$ and $\overline{\mu}$. The search is stopped when the two extreme coincides. It can easily be shown that this algorithm converges to the optimum.

6.3 External Modules

As shown in Figure 6.5, the distributions of the computation and inter–arrival times, $\mathcal{C}_i(c)$ and $\mathcal{Z}_i(c)$, can be extracted by analysing the execution traces of the task. As an example, the external tool TrcUtils [7] can be used for this purpose.

The tool uses the Linux `ftrace` functionality to capture such traces and is organised as a pipeline of trace filters: the first stage of the pipeline (the import filter) transforms the text traces generated by `ftrace`, which contain redundant information, into an internal binary format, which only contains the relevant information and can be used by later stages of the pipeline.

The next stages of the TrcUtils pipeline consist of a second set of filters that export traces in different formats, parse the internal format to gather various statistics about tasks execution, display the schedule, etc. In this context, the interesting functionality is the generation of PMFs of the computation and inter–arrival times, which can be exported by a new filter into the PROSIT format.

Currently, this mechanism operates correctly only for non self–suspending tasks (that is, real–time tasks for which a job never blocks, and the task blocks only at the end of each job). Work is being done to overcome this limitation taking inspiration from other techniques [32].

Another important activity supported by external tools is the generation of the mapping between probabilistic deadlines and the applications' quality. Work is in progress to use the PSNRTools [62] to evaluate the quality of media processing tasks, based on the PROSIT outputs.

PSNRTools is a set of video processing tools that can encode an original video stream according to some specified parameters, using `ffmpeg`, and remove some parts of the video that have been lost or corrupted by some processing application to generate an output stream. A list of lost frames can be generated, for example, by considering the probabilistic deadlines computed by PROSIT.

PSNRTools can then evaluate the differences between the output stream and the original uncompressed video stream, by computing the PSNR or SSIM between them; when a frame is lost, the quality index is computed by using the latest correct output frame (a behaviour similar to the one of a real player).

6.4 Experimental Validation

In order to show the applicability of PROSIT, we have considered 3 different scenarios characterising the main functionality provided by the tool. These scenarios comprise real–time tasks described by both synthetic and experimentally obtained distributions. All the results presented in this section have been measured on a Dell Precision T1700 equipped with an Intel Core i7 8–core processor operated at 3.6 Ghz and with 32 GB of RAM; the tool was compiled with a gcc 4.8.4 and `-O3` optimisation switch.

Moreover, in order to collect statistics of the time taken by PROSIT to perform the stochastic analysis or the budget optimisation, each scenario was analysed 50 times. Therefore, the computation times required to obtain these results, and reported here, correspond to the average time of the individual trials with its corresponding 95% confidence interval.

Additionally, PROSIT was used in [84] for the experimental validation and the probabilistic quality optimisation, in [98] for the stochastic analysis, and in [98, 6] for the estimation of the parameters of the model. PROSIT is freely available online[2] for the real–time systems research community and the industrial practitioners.

6.4.1 Analysis of Resource Reservation Scheduling

The first scenario corresponds to the solution of the analysis problem for a reservation–based scheduling algorithm. This scenario comprises two periodic real–time applications that are scheduled through a reservation–based scheduler and whose computation time is described by synthetic distributions. The first task is characterised by a computation time described by an independent and identically distributed random variable; on the other hand, the second task is characterised by a computation time described by a Markov Computation Time Model (MCTM) [98].

[2] https://bitbucket.org/luigipalopoli/prositool.git

Table 6.1: Probability of respecting a deadline equal to the task period with an assigned bandwidth of 51% for different choices of probability solvers. Additionally, in the second column, the time required by PROSIT to obtain these results. The probabilities are very close among each other, however the time required for the `analytic` method is 3 orders of magnitude smaller than the other methods.

Solver	Probability	Time
`analytic`	0.90541422	609.74 ± 9.28 μs
`cyclic`	0.92896863	495.66 ± 0.85 ms
`logarithmic`	0.92896868	471.18 ± 4.19 ms
`companion`	0.92896863	4739.13 ± 4.37 ms

The first task, τ_1, represents a periodic task with period $T_1 = 100000\,\mu$s and scheduling parameters: server period $T_1^s = 25000\,\mu$s and budget $Q_1^s = 12750\,\mu$s. The computation time is assumed to be independent and identically distributed, according to a beta distribution: $\mathcal{C}_i(c) = J(\alpha,\,\beta)c^{\alpha-1}\,(1-c)^{\beta-1}$, with support $c \in [13000,\,76500]$ μs, with $\alpha = 1.5$ and $\beta = 4$, and $J(\alpha,\,\beta)$ is a normalisation constant. The beta distribution is interesting because it is unimodal and has a finite support, which make it a good fit to approximate the behaviour of a large number of real–time applications.

We report the results of the comparison between the numeric solution resulting from the Cyclic Reduction algorithm [19] (`cyclic`), the Logarithmic Reduction algorithm [66] (`logarithmic`), the Companion Form algorithm [84] (`companion`) and the approximated Analytic Bound [84] (`analytic`). Table 6.1 shows the results obtained with PROSIT. These results were obtained with $\Delta = Q_1^s = 12750\,\mu$s for the `analytic` solver; while for the other solvers, $\Delta = 250\,\mu$s.

In accordance with our previous results [84], the `cyclic`, `logarithmic` and `companion` solvers produce almost the same result in terms of probability. Indeed, it is possible to observe that the differences start from the sixth digit. In the case of `analytic`, being this method an approximated bound, the probability computed is very close to the ones obtained by the numeric algorithms. However, the time required by PROSIT to obtain this result is several orders of magnitude below than the time required to obtain the numeric solutions.

For the second task, we decided to present the whole process depicted in Figure 6.9. Therefore, we have considered a periodic task with computation time described by a MCTM whose distributions were synthetically generated in simulation from a known HMM.

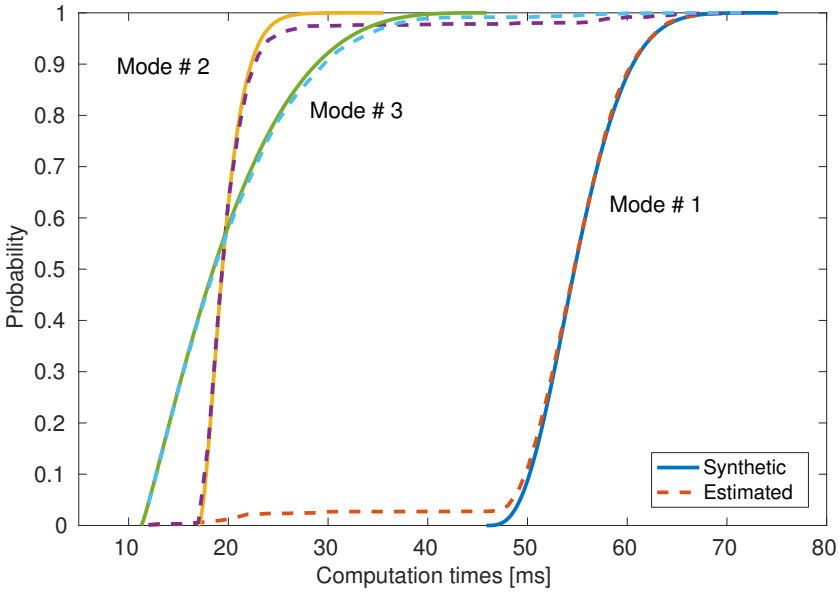

Figure 6.10: Cumulative distribution functions for the 3 different modes identified by PROSIT. The known synthetic distributions are represented in solid lines, while the distributions estimated by PROSIT are indicated in dashed lines.

This HMM is considered to have 3 states. The cumulative distribution function for each operating mode is shown, in solid lines, in Figure 6.10 and the transition probability matrix is given by:

$$M = \begin{bmatrix} 0.6666 & 0.1221 & 0.2113 \\ 0.2471 & 0.6452 & 0.1077 \\ 0.1315 & 0.0219 & 0.8466 \end{bmatrix}.$$

The PROSIT tool was fed with different sequences of synthetically generated samples, obtained from the previously described HMM. This process has been repeated multiple times, for different numbers of operating modes (ranging from 1 to 6). In order to speed up the process of parameter estimation, the generated samples were resampled with $\Delta = 1000\,\mu$s and the Baum–Welch algorithm was set to perform 100 iterations in the estimation process. The algorithm consistently identified 3 different modes, which present independence in the computation times associated to them, as shown in Table 6.2.

In particular, the transition probability matrix $\widehat{M}$ estimated with 15000 measures is

$$\widehat{M} = \begin{bmatrix} 0.6843 & 0.1150 & 0.2007 \\ 0.2217 & 0.6741 & 0.1042 \\ 0.1247 & 0.0228 & 0.8525 \end{bmatrix},$$

which has an error that is less than 2%. The estimated probability distributions for the computation times of each operating mode is presented, in dashed lines, in Figure 6.10. It

Table 6.2: Results of the numerical independence tests for the 3 sub–sequences estimated by the Baum–Welch algorithm. A p–value greater than 0.01 allows us to accept the independence assumption.

State	z–statistic	p–value
1	−0.3731	0.3546
2	0.5473	0.7079
3	0.1692	0.5672

is possible to observe that the estimated distributions look pretty similar when compared with the original ones. The measured computation time required by PROSIT to estimate the MCTM parameters was 97.05 ± 1.96 seconds.

As mentioned before, the second task, τ_2, whose computation time is described by a MCTM, represents a periodic task with period $T_2 = 100000\,\mu$s and scheduled through a CPU reservation with server period $T_2^s = 25000\,\mu$s and budget $Q_2^s = 13000\,\mu$s.

Figure 6.11 compares the probability of respecting the deadline estimated by the PROSIT tool when using the MCTM model with 3–states, the simulation results obtained with the simulator included in PROSIT and the simplified i.i.d model presented in [84]. Once again, the i.i.d. approximation produces an *optimistic* probability that could lead to problems in a real environment. Finally, the time taken by PROSIT to perform this stochastic analysis was 163.19 ± 1.08 ms.

6.4.2 Analysis of Fixed–Priority Scheduling

The second scenario corresponds to the stochastic analysis of periodic real–time tasks scheduled with a fixed–priority scheduling algorithm. PROSIT implements the stochastic analysis proposed in [38]. Hence, in order to test the results provided by our tool, we have used the same task sets presented in [50, 38], which are shown in Table 6.3.

The computation time of each task is given by a uniform distribution defined in the bounded set $\left\{c_i^{\min}, \ldots, c_i^{\max}\right\}$, where the symbols $c_i^{\min}$ and $c_i^{\max}$ denote the minimum and maximum computation time for the task τ_i. Additionally, all three task sets in Table 6.3 are assumed to be scheduled by a Rate Monotonic scheduling algorithm. These task sets correspond to synthetic systems and the authors claim that the parameters of the distributions were carefully chosen to accentuate the potential for missed deadlines.

The results obtained with the PROSIT tool are presented in Table 6.4. These results show the probability of respecting the deadline, expressed as the ratio between the number of jobs that finished before the deadline over the total number of jobs. The `simulation`

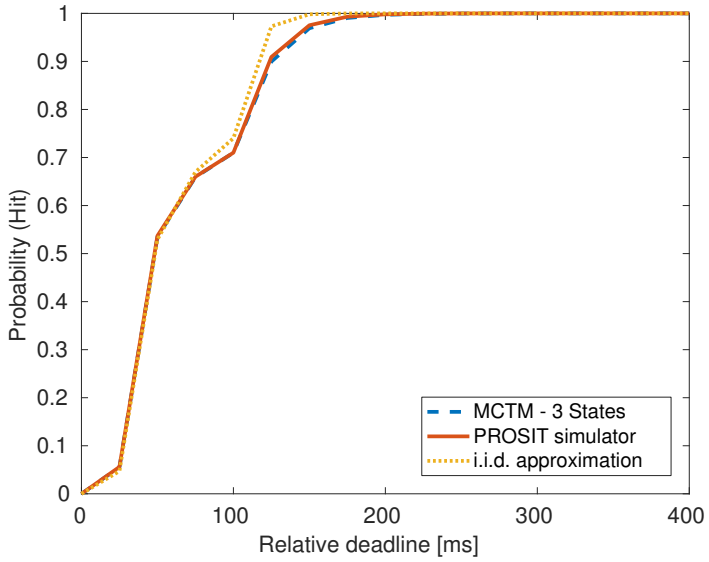

Figure 6.11: Comparison of the probability of respecting the deadline for different analysis methods, with a bandwidth fixed to 52%. From 75 ms onwards, the i.i.d. approximation produces too optimistic results when compared with the simulated ones. Modelling the computation times with a MCTM significantly reduces such optimism and the results from the analysis are more accurate with respect to the simulated ones.

results are copied from [50], while the `exact analysis` results are obtained from [38].

As reported for the previous scenario, the measured computation time required by PROSIT to obtain these results was 174.91 $\pm$ 0.27 seconds.

6.4.3 The Synthesis Problem

The third scenario corresponds to the solution of the synthesis problem, namely to find an optimal allocation of the bandwidth between the different tasks. In this case, we have evaluated the tool with 2 different task sets. The first task set comprises four periodic real–time applications that are scheduled through a reservation–based scheduler and whose computation times are described by synthetic distributions.

In Table 6.5, we report for each task: the period (T_i), the server period (T_i^s), the lower bound for the Quality of Service ($\underline{\mu}_i$), and the QoS function. The Quality is, in this example, a very simple function of the probability p of meeting a deadline set equal to the period. Specifically, the quality function used is defined as follows:

Table 6.3: Characterisation of the 3 task sets used for the stochastic analysis of periodic real–time tasks scheduled by a fixed–priority algorithm. The priorities are assigned according to Rate Monotonic (smaller period, higher priority). The computation times are described by an uniform distribution in the range $\left[c_i^{\min}, c_i^{\max}\right]$. These task sets were originally presented in [50].

Task Set Description						
Set	Task	T_i	D_i	$c_i^{\min}$	$\overline{c}_i$	$c_i^{\max}$
S_1	τ_1	300	300	72	100	128
	τ_2	400	400	72	150	228
S_2	τ_1	300	300	50	100	150
	τ_2	400	400	50	150	250
S_3	τ_1	300	300	1	100	199
	τ_2	400	400	1	150	299

$$\mathbf{linear}_{p_{min},p_{max},\alpha}(p) = \begin{cases} 0 & \text{if } p \leq p_{min} \\ \alpha\,(p - p_{min}) & \text{if } p_{min} < p < p_{max} \\ \alpha\,(p_{max} - p_{min}) & \text{if } p \geq p_{max} \end{cases}. \tag{6.3}$$

We have executed the synthesis procedure using the infinity norm as the global cost function. This procedure was done using as solvers both the approximated Analytic Bound [84] (`analytic`) and the numeric solution resulting from the Cyclic Reduction algorithm [19] (`cyclic`). The former produces a conservative bound on the probability of meeting the deadline. Thereby, an optimisation algorithm based on the analytic bound simply produces suboptimal solutions. On the contrary, the `cyclic` solver produces an exact solution for the probability (within the limits of a numeric solution) and hence the application of the optimisation algorithm described in the previous section produces the optimal solution.

The results of the optimisation using the analytic bound and the CR are reported in Table 6.6. In particular, we report the optimal value of the budget, the corresponding probability of meeting the deadline and the value of the quality function obtained from Equation 6.3.

In order to make a fair comparison, we have re–evaluated the probability for the optimal budgets using an exact solver (CR) also for the analytic solution. The CR solution clearly produces a closer approximation of the optimal as compared to the suboptimal produced by the analytic solution. Indeed, the infinity norm of the quality is 0.39 for CR and 0.32 for the analytic bound. But, the computation time was 3.74 $\pm$ 0.02 ms for the

Table 6.4: The probability of respecting a deadline equal to the task period for the task sets presented in Table 6.3. The results labelled "Simulation" and "Exact Analysis" were taken from the original papers presenting those methods [50] and [38] respectively.

Set	Task	Simulation	Exact Analysis	PROSIT
S_1	τ_1	100.0 ± 0.0	100.0	100.0
	τ_2	95.3 ± 0.1	95.3	95.3
S_2	τ_1	100.0 ± 0.0	100.0	100.0
	τ_2	92.6 ± 0.2	92.6	92.6
S_3	τ_1	100.0 ± 0.0	100.0	99.9
	τ_2	80.8 ± 0.1	80.8	80.8

Table 6.5: Characterisation of the synthetic task set used for the solution of the optimisation problem. The minimum Quality of Service required is denoted by μ_i; while the parameters of the Quality of Service function are described in Equation (6.3).

Task Description				
Task	T_i	T_i^s	$\underline{\mu}_i$	Quality function
τ_1	400	200	0.1	$\text{linear}_{0.01,\,0.95,\,0.5}$
τ_2	600	100	0.1	$\text{linear}_{0.01,\,0.85,\,0.5}$
τ_3	300	100	0.1	$\text{linear}_{0.01,\,0.95,\,0.5}$
τ_4	300	100	0.1	$\text{linear}_{0.01,\,0.95,\,0.5}$

analytic bound and 1042.99 ± 0.62 seconds for CR.

The second task set correspond to a real application, where a computing board is used to process, in real–time, multiple videos at the same time. It is based on two different videos encoded with a bit–rate of 600 Kb/s: the first one, "BridgeClose"[3], displays a bridge with occasional people coming through, hence it is characterised by a single, almost static scene with slow movements; the second video, "The UFO"[4], is a movie trailer characterised by frequent scene changes and rapid movements.

One of the best known ways to evaluate the quality of a video is the Peak Signal–to–Noise Ratio (PSNR), which is computed comparing pairwise the frames of the original raw video and of the one obtained after encoding and decoding it [63, 62]. This metric can be evaluated considering a video player implemented as a periodic real–time task.

[3] http://trace.eas.asu.edu/yuv/index.html

[4] http://www.theufo.net/index2.htm – trailer 1

Table 6.6: Optimal budget allocation for the tasks in the synthetic task set of the optimisation problem. For each method it is reported: the assigned bandwidth, the probability of respecting the deadline and the Quality of Service metric. The assigned bandwidths are almost the same for the 2 methods.

Task	`analytic`			`cyclic`		
	Q_i^s	p_i	μ	Q_i^s	p_i	μ
τ_1	35	0.691448	0.3407	34	0.788359	0.3892
τ_2	25	0.644236	0.3171	25	0.830756	0.4104
τ_3	29	0.709177	0.3496	29	0.819039	0.4045
τ_4	24	0.681572	0.3358	24	0.825144	0.4076

Table 6.7: Experimental results of the probabilistic optimisation of the Video Decoder Application for the two streams: BridgeClose and The UFO. For each method it is reported: the assigned bandwidth, the probability of respecting the deadline and the Quality of Service metric.

Task	`analytic`			`cyclic`		
	Q_i^s	p_i	μ	Q_i^s	p_i	μ
BridgeClose	2802	0.738281	39.6030	2550	0.746905	39.6799
The UFO	6684	0.997792	41.5377	6949	0.998912	41.5847

If a job misses its deadline, the video frame is not played back but it is decoded (to allow the incremental decoding of the frames that follow). In this case, the behaviour of most players is to fill–in the "hole" by simply repeating the last decoded frame. This is perceived by the user as a reduction in quality, which is well reflected in a degradation of the PSNR.

The PSNR was interpolated by a line with a slope of 8.9128 and a y–intercept of 33.112 for "BridgeClose" and a slope of 42.051 for "The UFO". This difference is explained by the different nature of the movies: static the former and dynamic the latter. Both movies were decoded using a player executed by a periodic task and scheduled by the SCHED_DEADLINE policy. The distributions of the computation times were recorded on a notebook powered by an Intel Atom Processor, and the resulting CDFs are shown in Figure 6.12.

Choosing 30 ms for the activation period (corresponding to 33 fps), setting the server period to 10 ms, and restricting the total bandwidth available to 95% (to leave some room for other applications), the tool produces the results presented in Table 6.7. We identified

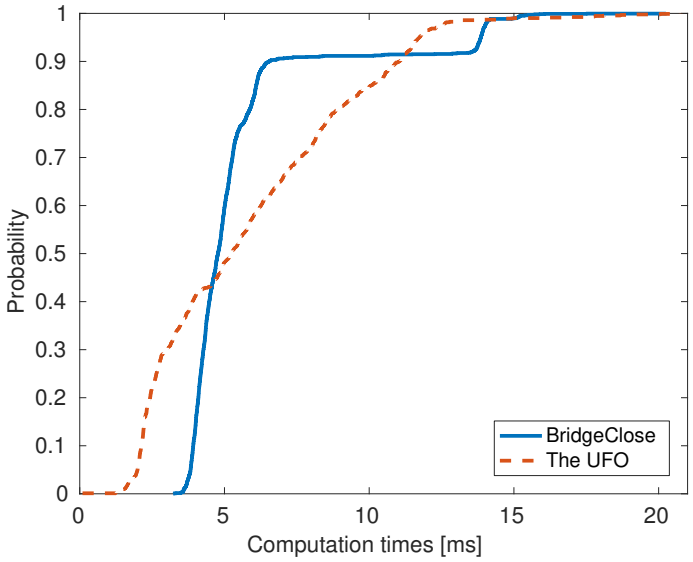

Figure 6.12: Empirical cumulative distribution functions for the computation times experimentally obtained (measured) from the Video Decoder application for the two streams: BridgeClose and The UFO.

empirically the minimum acceptable PSNR as 31 for "BridgeClose" and 39 for "Ufo". These values were codified as constraints in the optimisation problem. As a solver for the probability computation we have considered the approximated Analytic Bound [84] (`analytic`) with $\Delta = Q_i^s$ and the numeric solution resulting from the Cyclic Reduction algorithm [19] (`cyclic`) with $\Delta = 50\,\mu$s.

For both solvers, the solution assigns a larger bandwidth to the "The UFO" stream, approximately 69% for the `cyclic` and 66% for the `analytic`; this is because the quality degrades more quickly with the probability of meeting the deadline for "The UFO" than for "BridgeClose". In this case, the use of the analytic bound produces an optimal value of 41.5377 in 120.38 ± 5.6 ms, which is pretty much the same value as the one obtained with the cyclic reduction (41.5847), however the computation time of 510.28 ± 0.21 seconds for the latter is three orders of magnitude above. This result confirms our preliminary observation, the analytic approach is viable if a quick computation is required (e.g., if the tool is used on–line), while the exact approaches are preferable when an offline execution of the tool allows for a more precise solution.

6.4.4 PROSIT Scalability

As mentioned in Section 5.5.3, the tractability of the system depends on the dimension of the probability transition matrix P of the QBDP presented in Equation 5.8 and described in Section 5.3.1. The dimension of the transition matrix depends on the `granularity`

Table 6.8: Dimension of the probability transition matrix of the QBDP for two different applications. In the left, the 3D map navigator application presented in Section 4.3.1 and scheduled using a resource reservation algorithm. In the right, the task set $S1$ presented in Table 6.3 and scheduled using a fixed–priority algorithm.

Granularity [μs]	Matrix size	Hyperperiod [ms]	Matrix size
24000	34×34	300	426×426
12000	64×64	400	538×538
6000	126×126	600	962×962
3000	250×250	1200	1394×1394
1500	498×498	1400	2034×2034
1000	748×748	1500	2234×2234
500	1494×1494	1750	2346×2346
250	2986×2986	2000	2458×2458
100	7464×7464	2100	2658×2658

parameter (the scaling factor Δ).

Additionally, depending on the scheduling algorithm, the dimension of the transition matrix could be affected by: the number of modes identified for the MCTM, the task period (T) and the scheduling parameters (Q^s, T^s) for the case of resource reservations; while for the case of fixed–priorities it can be affected by the hyperperiod of the tasks.

Table 6.8 shows the dimension of the probability transition matrix for different values of the `granularity` parameter for the 3D map navigator application example, presented in Section 4.3.1. For the case of fixed–priority scheduling, it is presented the task set $S1$ described in Table 6.3 with a `granularity` $= 1000\mu$s and different values for the hyperperiod, meaning different choices for the task period $T_1 \in \{200, 250, 300\}$ and $T_2 \in \{300, 350, 400\}$.

Chapter 7

Conclusions and Future Work

Extensive literature supports the paradigm of stochastic analysis for real–time systems under the assumption that the stochastic process describing the computation times is independent and identically distributed (i.i.d.). This thesis presents some of the most relevant and conventional solutions that have been explored in the scientific community.

A frequent choice is to ignore the correlation structure and to consider the process as i.i.d., which we have shown with experimental data that may fall short of producing decent results in some applications.

Moreover, the ever increasing development of real–time robotics applications presenting correlation in their computation times has raised the attention of the community to the proper applicability of the i.i.d. assumption when dealing with this type of applications.

This dissertation elaborates on the identification of a particular model for the computation times that are not independent and identically distributed. In particular, a more general Markovian model, that we have called: the "Markov Computation Time Model" (MCTM), is introduced. This model associates a different state with every working condition of the system and generates a different distribution in each of these cases, recovering the i.i.d. process as a special case.

In order to apply the machinery developed for the probabilistic guarantees within the MCTM model, it is necessary to properly characterise its parameters. Hence, this research focused on an effective procedure for identifying the parameters of the MCTM from a limited number of observations, adapting ideas and techniques from HMM estimation. The effectiveness of the approach has been proved on both synthetic data and on real data sets taken from different robotic applications.

When the identification process determines that the computation times of the given real–time application fulfil the conditions to be described as a MCTM, it would be possible to perform the stochastic analysis. Therefore, we have also presented an adaptation of the standard techniques on probabilistic guarantees to cope with this new type of model. The

adapted technique provides numerically efficient probabilistic guarantees for respecting the deadlines.

Once again, through a series of experimental cases, we have proved that the proposed methods can be considered a more accurate and suitable solution for the stochastic analysis. The obtained results both in synthetic and real–life scenarios produced a more tight condition when compared with the one obtained by the customary i.i.d. assumption.

Moreover, we have developed a software tool for the probabilistic design of real–time systems. The tool, called PROSIT, is implemented as a C++ library under the GNU GPL license and is available, with several examples and datasets including all the experiments presented in this thesis, at `https://bitbucket.org/luigipalopoli/prositool.git`.

Various research activities can be organized from this point. First, it is possible to extend the analysis presented in this thesis to the case of aperiodic tasks, which are more frequently encountered in robotic applications. Probabilistic guarantees for aperiodic tasks can be given in case of i.i.d. computation and inter–arrival times, as shown in Chapter 4. If the stochastic process describing the inter–arrival times can be modelled as a Markov Modulated Process, it could be possible to apply a similar technique in order to estimate its parameters and the stochastic analysis could also be extended to cope with Markovian inter–arrival times.

Future research efforts can also be directed towards the development of a supervisor algorithm that periodically verifies whether the modelled system properly represents the current situation. For instance, considering a mobile robot which navigates a certain environment, it is perfectly possible that the estimated parameters of the MCTM vary in time (e.g., the robot faces an environment which is modelled by a 3–states MCTM while suddenly enters a different environment which is better modelled by a 6–states MCTM).

Depending on different factors such as: the minimum number of samples required to perform the parameter estimation, the time required to estimate the parameters of the MCTM, the computational power required by the robot to performs the analysis; it could be possible to devise an online and iterative algorithm which will incrementally adapt the estimated parameters of the model as the samples are collected.

Such a method could potentially pave the way for the development of adaptive scheduling schemes: adaptive reservations, in which the scheduling parameter, namely the bandwidth assigned to each task is adjusted online in order for the application to meet a target on the Quality of Service requirements (e.g., probability of respecting the deadline).

Another research avenue regards modern multi–core systems. Although the proposed technique focuses on single–processor systems, it can also be applied on multi–core architectures with partitioned scheduling. Statically assigning a set of tasks to a cpuset containing a single CPU simulates, a single–processor architecture. Under these condi-

tions it is possible to perform the stochastic analysis of the tasks as if they were executed on a single–processor system.

However it is necessary to investigate whether the measured computation times obtained from a single–processor system are comparable to the ones obtained in a multi–processor system with partitioned scheduling in order to characterise and account for the overhead introduced by the architecture.

Additionally, it could be possible to combine the proposed techniques in order to produce a comprehensive strategy for the deployment of the real–time tasks. The PROSIT tool could include some extra features allowing the designer to analyse the schedulability of a set of tasks running on a multi–processor architecture with partitioned scheduling.

The extra features should include a CPU load balancing algorithm in such a way that, when possible, the tool will also produce the optimal partition of the tasks among the defined cpusets guaranteeing a fair load among the CPUs and the respect of the required Quality of Service metrics.

Another possible direction to follow, based on the proposed approach developed in this thesis, considers the investigation of the level of sensitivity/tolerance provided by the presented analysis regarding the respect of the measured computation times towards the empirical PMF.

This study could be considerably useful in the case of persistent effects derived of interference between tasks. In particular, any possible additional time required by a cache miss or a bus contention introduces more variability in the computation times. If not accounted for, this variability could lead to severe differences between the probability of respecting the deadline obtained by the system and the one estimated by the analysis.

Regarding the PROSIT tool, an intense development activity covering a significant number of features of the tool, such as: 1) the definition of new solution algorithms for probability computation, 2) the definition of new quality metrics, 3) the definition of different optimisation algorithms, 4) the definition of a library for different distribution types (e.g., Uniform, Beta, Gaussian, etc.) allowing the user to customise the distribution parameters, 5) full support for fixed–priority systems including the solution for the synthesis problem by following the algorithms presented in [77], is currently ongoing.

From the modelling point of view, we are looking at different types of real–time applications (e.g., multi–task and distributed applications); the analysis of such applications will extend the library of available quality functions. Finally, the production of a Graphical User Interface (GUI) is also planned as an important future activity.

List of Tables

List of Figures

Bibliography

[1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real–time systems. In *Proceedings of the 19th IEEE Real–Time Systems Symposium*, pages 4–13, Madrid, Spain, December 1998. IEEE.

[2] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the 6th International Conference on Real–Time Computing Systems and Applications*, pages 70–77, Hong Kong, China, December 1999. IEEE.

[3] Luca Abeni and Giorgio Buttazzo. QoS guarantee using probabilistic dealines. In *Proceedings of the 11th Euromicro Conference on Real–Time Systems*, pages 242–249, York, England, June 1999. IEEE.

[4] Luca Abeni and Giorgio Buttazzo. Stochastic analysis of a reservation–based system. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pages 946–952, San Francisco, USA, April 2001. IEEE.

[5] Luca Abeni, Daniele Fontanelli, and Luigi Palopoli. Application of the Quasi–Birth–Death Processes techniques to probablistic guarantees of soft real–time systems scheduled by resource reservations. URL: `http://disi.unitn.it/~palopoli/publications/QBDP-TR.pdf`, 2015. Accessed: 2018-04-27.

[6] Luca Abeni, Daniele Fontanelli, Luigi Palopoli, and Bernardo Villalba Frías. A markovian model for the computation time of real–time applications. In *Proceedings of the International Instrumentation and Measurement Technology Conference*, pages 1–6, Turin, Italy, May 2017. IEEE.

[7] Luca Abeni and Nicola Manica. Interoperable tracing tools. In *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real–Time Systems*, pages 33–38, Pisa, Italy, July 2012.

[8] Luca Abeni, Nicola Manica, and Luigi Palopoli. Efficient and robust probabilistic guarantees for real–time tasks. *Journal of Systems and Software*, 85(5):1147–1156, 2012.

[9] Sebastian Altmeyer and Robert I Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *Conference and Exhibition on Design, Automation and Test in Europe*, pages 1–6, Dresden, Germany, March 2014. IEEE.

[10] James Anderson, Sanjoy Baruah, and Björn Brandenburg. Multicore operating–system support for mixed criticality. In *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, volume 4, page 7, San Francisco, USA, April 2009. IEEE.

[11] Alia Atlas and Azer Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the 19th IEEE Real–Time Systems Symposium*, pages 123–132, Madrid, Spain, December 1998. IEEE.

[12] Leonard Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov process. *Inequalities*, 3:1–8, 1972.

[13] Guillem Bernat, Alan Burns, and Martin Newby. Probabilistic timing analysis: An approach using copulas. *Journal of Embedded Computing*, 1(2):179–194, 2005.

[14] Guillem Bernat, Antoine Colin, and Stefan Petters. WCET analysis of probabilistic hard real–time systems. In *Proceedings of the 23rd IEEE Real–Time Systems Symposium*, pages 279–288, Austin, USA, December 2002. IEEE.

[15] Guillem Bernat, Antoine Colin, and Stefan Petters. PWCET: A tool for probabilistic worst–case execution time analysis of real–time systems. URL: `https://www.cs.york.ac.uk/ftpdir/reports/2003/YCS/353/YCS-2003-353.pdf`, 2003. Accessed: 2018-04-27.

[16] Paolo Bevilacqua, Marco Frego, Enrico Bertolazzi, Daniele Fontanelli, Luigi Palopoli, and Francesco Biral. Path planning maximising human comfort for assistive robots. In *Conference on Control Applications*, pages 1421–1427, Buenos Aires, Argentina, September 2016. IEEE.

[17] Paolo Bevilacqua, Marco Frego, Daniele Fontanelli, and Luigi Palopoli. Reactive planning for assistive robots. *Robotics and Automation Letters*, 3(2):1276–1283, 2018.

[18] Dario Bini, Guy Latouche, and Beatrice Meini. *Numerical methods for structured Markov chains*. Oxford University Press, 2005.

[19] Dario Bini and Beatrice Meini. On cyclic reduction applied to a class of toeplitz-like matrices arising in queueing problems. In *Computations with Markov Chains*, pages 21–38. Springer, 1995.

[20] Dario Bini, Beatrice Meini, Sergio Steffé, Juan Pérez, and Benny Van Houdt. SMCSolver and Q–MAM: tools for matrix–analytic methods. *ACM SIGMETRICS Performance Evaluation Review*, 39(4):46–46, 2012.

[21] Enrico Bini and Giorgio Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[22] Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. Rate monotonic analysis: the hyperbolic bound. *IEEE Transactions on Computers*, 52(7):933–942, 2003.

[23] Enrico Bini, Thi Huyen Châu Nguyen, Pascal Richard, and Sanjoy Baruah. A response–time bound in fixed–priority scheduling with arbitrary deadlines. *IEEE Transactions on Computers*, 58(2):279–286, 2009.

[24] Giorgio Buttazzo. *Hard real–time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer, 2011.

[25] Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in hidden Markov models (Springer Series in Statistics)*. Springer, 2005.

[26] Christos Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer, 2006.

[27] Anton Cervin, Bo Lincoln, Johan Eker, Karl–Erik Arzén, and Giorgio Buttazzo. The jitter margin and its application in the design of real–time control systems. In *Proceedings of the 10th International Conference on Real–Time and Embedded Computing Systems and Applications*, pages 1–9, Gothenburg, Sweden, August 2004. IEEE.

[28] Younès Chandarli, Frédéric Fauberteau, Damien Masson, Serge Midonnet, and Manar Qamhieh. YARTISS: A tool to visualize, test, compare and evaluate real–time scheduling algorithms. In *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real–Time Systems*, pages 21–26, Pisa, Italy, July 2012.

[29] Maxime Chéramy, Pierre–Emmanuel Hladik, and Anne–Marie Déplanche. SimSo: A simulation tool to evaluate real–time multiprocessor scheduling algorithms. In *Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real–Time Systems*, pages 37–42, Madrid, Spain, July 2014.

[30] Alessio Colombo, Daniele Fontanelli, Axel Legay, Luigi Palopoli, and Sean Sedwards. Efficient customisable dynamic motion planning for assistive robots in complex human environments. *Journal of ambient intelligence and smart environments*, 7(5):617–634, 2015.

[31] Tommaso Cucinotta, Luca Abeni, Giuseppe Lipari, Luca Marzario, and Luigi Palopoli. QoS management through adaptive reservations. *Real–Time Systems Journal*, 29(2–3):131–155, 2005.

[32] Tommaso Cucinotta, Fabio Checconi, Luca Abeni, and Luigi Palopoli. Self–tuning schedulers for legacy real–time applications. In *Proceedings of the 5th European Conference on Computer Systems*, pages 55–68, Paris, France, April 2010. ACM.

[33] Tommaso Cucinotta, Luigi Palopoli, and Luca Marzario. Stochastic feedback–based control of QoS in soft real–time systems. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, volume 4, pages 3533–3538, Nassau, Bahamas, December 2004. IEEE.

[34] Tommaso Cucinotta, Luigi Palopoli, Luca Marzario, Giuseppe Lipari, and Luca Abeni. Adaptive reservations in a Linux environment. In *Proceedings of the 10th IEEE Real–Time and Embedded Technology and Applications Symposium*, pages 238–245, Toronto, Canada, May 2004. IEEE.

[35] Liliana Cucu–Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco Cazorla. Measurement-based probabilistic timing analysis for multi–path programs. In *Proceedings of the 24th Euromicro Conference on Real–Time Systems*, pages 91–101, Pisa, Italy, July 2012. IEEE.

[36] Liliana Cucu–Grosjean and Eduardo Tovar. A framework for the response time analysis of fixed–priority tasks with stochastic inter–arrival times. *ACM SIGBED Review*, 3(1):7–12, 2006.

[37] Pierre Devijver. Baum's forward–backward algorithm revisited. *Pattern Recognition Letters*, 3(6):369–373, 1985.

[38] José Díaz, Daniel García, Kanghee Kim, Chang–Gun Lee, Lucia Lo Bello, José López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real–time systems. In *Proceedings of the 23rd IEEE Real–Time Systems Symposium*, pages 289–300, Austin, USA, December 2002. IEEE.

[39] José Díaz, José López, Manuel García, Antonio Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *Proceedings of the 25th IEEE Real–Time Systems Symposium*, pages 197–207, Lisbon, Portugal, December 2004. IEEE.

[40] Sean Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

[41] Stewart Edgar and Alan Burns. Statistical analysis of wcet for scheduling. In *Proceedings of the 22nd IEEE Real–Time Systems Symposium*, pages 215–224, London, United Kingdom, December 2001. IEEE.

[42] Francesco Farina, Daniele Fontanelli, Andrea Garulli, Antonio Giannitrapani, and Domenico Prattichizzo. Walking ahead: The headed social force model. *PloS one*, 12(1):1–23, 2017.

[43] Wolfgang Fischer and Kathleen Meier–Hellstern. The markov–modulated poisson process (MMPP) cookbook. *Performance evaluation*, 18(2):149–171, 1993.

[44] Martin Fischler and Robert Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[45] Daniele Fontanelli, Luca Greco, and Luigi Palopoli. Soft real–time scheduling for embedded control systems. *Automatica*, 49(8):2330–2338, 2013.

[46] Daniele Fontanelli, Federico Moro, Tizar Rizano, and Luigi Palopoli. Vision–based robust path reconstruction for robot control. *IEEE Transactions on Instrumentation and Measurement*, 63(4):826–837, 2014.

[47] Danile Fontanelli, Luigi Palopoli, and Luca Abeni. The continuous stream model of computation for real–time control. In *Proceedings of the 34th IEEE Real–Time Systems Symposium*, pages 150–159, Vancouver, Canada, December 2013. IEEE.

[48] David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[49] Mark Gardner and Jane Liu. Analyzing stochastic fixed–priority real–time systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 44–58, Amsterdam, The Netherlands, March 1999. Springer.

[50] Mark Gardner and Jane Liu. *Probabilistic analysis and scheduling of critical soft real–time systems*. University of Illinois at Urbana–Champaign, 1999.

[51] Michael González, José Gutiérrez, José Palencia, and José Drake. MAST: Modeling and analysis suite for real–time applications. In *Proceedings of the 13th Euromicro Conference on Real–Time Systems*, pages 125–134, Delft, The Netherlands, June 2001. IEEE.

[52] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. URL: `http://eigen.tuxfamily.org`, 2010. Accessed: 2018-04-27.

[53] Fabrice Guet, Luca Santinelli, and Jérôme Morio. On the reliability of the probabilistic worst–case execution time estimates. In *Proceedings of the 8th European Congress on Embedded Real–Time Software and Systems*, pages 758–767, Toulouse, France, January 2016.

[54] Claude–Joachim Hamann, Jork Loser, Lars Reuther, Sebastian Schonberg, Jean Wolter, and Hermann Hartig. Quality-assuring scheduling-using stochastic behavior to improve resource utilization. In *Proceedings of the 22nd IEEE Real–Time Systems Symposium*, pages 119–128, London, United Kingdom, December 2001. IEEE.

[55] Mathai Joseph and Paritosh Pandya. Finding response times in a real–time system. *The Computer Journal*, 29(5):390–395, 1986.

[56] Giordano Kaczynski, Lucia Lo Bello, and Thomas Nolte. Deriving exact stochastic response times of periodic tasks in hybrid priority–driven soft real–time systems. In *Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation*, pages 101–110, Patras, Greece, September 2007. IEEE.

[57] Dong–In Kang, Richard Gerber, and Manas Saksena. Performance–based design of distributed real–time systems. In *Proceedings of the 3rd IEEE Real–Time Technology and Applications Symposium*, pages 2–13, Montreal, Canada, June 1997. IEEE.

[58] Sertac Karaman, Matthew Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the RRT*. In *Proceedings of the International Conference on Robotics and Automation*, pages 1478–1483, Shanghai, China, May 2011. IEEE.

[59] Samuel Karlin. *A first course in stochastic processes*. Academic press, 2014.

[60] Kanghee Kim, José Díaz, Lucia Lo Bello, José López, Chang–Gun Lee, and Sang Lyul Min. An exact stochastic analysis of priority–driven periodic real–time systems and its approximations. *IEEE Transactions on Computers*, 54(11):1460–1466, 2005.

[61] Namhoon Kim, Jeremy Erickson, and James Anderson. Mixed–criticality on multicore (MC2): A status report. In *Proceedings of the 10th Annual Workshop on Operating Systems Platforms for Embedded Real–Time Applications*, pages 45–50, Madrid, Spain, July 2014. IEEE.

[62] Csaba Kiraly, Luca Abeni, and Renato Lo Cigno. Effects of p2p streaming on video quality. In *Proceedings of the IEEE International Conference on Communications*, pages 1–5, Cape Town, South Africa, May 2010. IEEE.

[63] Jirka Klaue, Berthold Rathke, and Adam Wolisz. Evalvid – a framework for video transmission and quality evaluation. In *Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 255–272, Urbana, USA, September 2003. Springer.

[64] James Kuffner and Steven LaValle. RRT–connect: An efficient approach to single–query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001, San Francisco, USA, April 2000. IEEE.

[65] Solomon Kullback and Richard Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[66] Guy Latouche and Vaidyanathan Ramaswami. A logarithmic reduction algorithm for Quasi–Birth–Death processes. *Journal of Applied Probability*, 30(3):650–674, 1993.

[67] Guy Latouche and Vaidyanathan Ramaswami. *Introduction to matrix analytic methods in stochastic modeling*, volume 5. Society for Industrial Mathematics, 1999.

[68] John Lehoczky. Real–Time queueing theory. In *Proceedings of the 17th IEEE Real–Time Systems Symposium*, pages 186–195, Washington DC, USA, December 1996. IEEE.

[69] John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 10th IEEE Real–Time Systems Symposium*, pages 166–171, Santa Monica, USA, December 1989. IEEE.

[70] Juri Lelli, Claudio Scordino, Luca Abeni, and Dario Faggioli. Deadline scheduling in the Linux kernel. *Software: Practice and Experience*, 46(6):821–839, 2016.

[71] Benjamin Lesage, David Griffin, Sebastian Altmeyer, and Robert Davis. Static probabilistic timing analysis for multi–path programs. In *Real-Time Systems Symposium, 2015 IEEE*, pages 361–372, San Antonio, USA, December 2015. IEEE.

[72] Chung Laung Liu and James Layland. Scheduling algorithms for multiprogramming in a hard–real–time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[73] Rui Liu, Alex Mills, and James Anderson. Independence thresholds: Balancing tractability and practicality in soft real–time stochastic analysis. In *Proceedings of the 35th IEEE Real–Time Systems Symposium*, pages 314–323, Rome, Italy, December 2014. IEEE.

[74] Nicola Manica, Luca Abeni, and Luigi Palopoli. QoS support in the X11 window system. In *Proceedings of the 14th IEEE Real–Time and Embedded Technology and Applications Symposium*, pages 103–112, St. Louis, USA, April 2008. IEEE.

[75] Nicola Manica, Luigi Palopoli, and Luca Abeni. Numerically efficient probabilistic guarantees for resource reservations. In *Proceedings of the 17th IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–8, Krakow, Poland, September 2012. IEEE.

[76] Dorin Maxim and Antoine Bertout. Analysis and simulation tools for probabilistic real-time systems. In *Proceedings of the 8th International Workshop on Analysis Tools and Methodologies for Embedded and Real–Time Systems*, pages 1–6, Dubrovnik, Croatia, June 2017.

[77] Dorin Maxim, Olivier Buffet, Luca Santinelli, Liliana Cucu-Grosjean, and Robert Davis. Optimal priority assignment algorithms for probabilistic real–time systems. In *Proceedings of the 19th IEEE International Conference on Real–Time and Network Systems*, pages 129–138, Nantes, France, September 2011. IEEE.

[78] Dorin Maxim and Liliana Cucu–Grosjean. Response time analysis for fixed–priority tasks with multiple probabilistic parameters. In *Proceedings of the 34th IEEE Real–Time Systems Symposium*, pages 224–235, Vancouver, Canada, December 2013. IEEE.

[79] Alex Mills and James Anderson. A stochastic framework for multiprocessor soft real–time scheduling. In *Proceedings of the 16th IEEE Real–Time and Embedded Technology and Applications Symposium*, pages 311–320, Stockholm, Sweden, April 2010. IEEE.

[80] Alex Mills and James Anderson. A multiprocessor server–based scheduler for soft real–time tasks with stochastic execution demand. In *Proceedings of the 17th International Conference on Embedded and Real–Time Computing Systems and Applications*, pages 207–217, Toyama, Japan, August 2011. IEEE.

[81] Marcel Neuts. *Matrix–geometric solutions in stochastic models: An algorithmic approach*. Dover Publications, 1981.

[82] Luigi Palopoli, Antonis Argyros, Josef Birchbauer, Alessio Colombo, Daniele Fontanelli, Axel Legay, Andrea Garulli, Antonello Giannitrapani, David Macii, Federico Moro, Payam Nazemzadeh, Pashalis Padeleris, Roberto Passerone, Georg Poier, Domenico Prattichizzo, Tizar Rizano, Luca Rizzon, Stefano Scheggi, and Sean Sedwards. Navigation assistance and guidance of older adults across complex public spaces: The DALi approach. *Intelligent Service Robotics*, 8(2):77–92, 2015.

[83] Luigi Palopoli, Tommaso Cucinotta, and Antonio Bicchi. Quality of service control in soft real–time applications. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 1, pages 664–669, Maui, USA, December 2003. IEEE.

[84] Luigi Palopoli, Daniele Fontanelli, Luca Abeni, and Bernardo Villalba Frías. An analytical solution for probabilistic guarantees of reservation based soft real–time systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):640–653, 2016.

[85] Luigi Palopoli, Daniele Fontanelli, Nicola Manica, and Luca Abeni. An analytical bound for probabilistic deadlines. In *Proceedings of the 24th Euromicro Conference on Real–Time Systems*, pages 179–188, Pisa, Italy, July 2012. IEEE.

[86] Luigi Palopoli, Giuseppe Lipari, Gerardo Lamastra, Luca Abeni, Gabriele Bolognini, and Paolo Ancilotti. An object oriented tool for simulating real–time distributed control systems. *Software Practice and Experience*, 32(9):907–932, 2002.

[87] Athanasios Papoulis and Unnikrishna Pillai. *Probability, random variables, and stochastic processes*. Tata McGraw–Hill Education, 2002.

[88] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. A predictable execution model for cots–based embedded systems. In *Proceedings of the 17th IEEE Real–Time and Embedded Technology and Applications Symposium*, pages 269–279, April 2011.

[89] Lawrence Rabiner and Biing–Hwang Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[90] Ragunathan Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource Kernels: A resource–centric approach to real–time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, pages 150–165, January 1998.

[91] Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. A resource allocation model for QoS management. In *Proceedings of the 18th IEEE Real–Time Systems Symposium*, pages 298–307, San Francisco, USA, December 1997. IEEE.

[92] Luca Santinelli, Fabrice Guet, and Jerome Morio. Revising measurement–based probabilistic timing analysis. In *Proceedings of the 23rd IEEE Real–Time and Embedded Technology and Applications Symposium*, pages 199–208, Pittsburgh, USA, April 2017. IEEE.

[93] Marcelo Santos, Björn Lisper, George Lima, and Veronica Lima. Sequential composition of execution time distributions by convolution. In *Proceedings of the 4th Workshop on Compositional Theory and Technology for Real–Time Embedded Systems*, pages 30–37, Vienna, Austria, November 2011. University of York, Department of Computer Science.

[94] Takahiro Shinozaki. HMM state clustering based on efficient cross–validation. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, pages 1157–1161, Toulouse, France, May 2006. IEEE.

[95] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. Cheddar: a flexible real–time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.

[96] Too–Seng Tia, Zhong Deng, Mallikarjun Shankar, Matthew Storch, Jun Sun, L–C Wu, and Jane Liu. Probabilistic performance guarantee for real–time tasks with varying computation times. In *Proceedings of the IEEE Real–Time Technology and Applications Symposium*, pages 164–173, Chicago, USA, May 1995. IEEE.

[97] Richard Urunuela, Anne-Marie Déplanche, and Yvon Trinquet. STORM: a simulation tool for real–time multiprocessor scheduling evaluation. In *Proceedings of the 15th IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–8, Bilbao, Spain, September 2010. IEEE.

[98] Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. Probabilistic real–time guarantees: There is life beyond the iid assumption (outstanding paper). In *Proceedings of the 23rd IEEE Real–Time and Embedded Technology and Applications Symposium*, pages 175–186, Pittsburgh, USA, April 2017. IEEE.

[99] Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. The PROSIT tool: Towards the optimal design of probabilistic soft real–time systems. *Software: Practice and Experience*, 2018. Accepted for publication.

[100] Bryan Ward, Jonathan Herman, Christopher Kenna, and James Anderson. Making shared caches more predictable on multicore platforms (outstanding paper award). In *Proceedings of the 25th Euromicro Conference on Real–Time Systems*, pages 157–167, Paris, France, July 2013. IEEE.

[101] Gang Yao, Rodolfo Pellizzoni, Stanley Bak, Emiliano Betti, and Marco Caccamo. Memory–centric scheduling for multicore hard real–time systems. *Real–Time Systems Journal*, 48(6):681–715, 2012.